

Copyright  
by  
Sharayu Arun Moharir  
2014

The Dissertation Committee for Sharayu Arun Moharir  
certifies that this is the approved version of the following dissertation:

**Resource Allocation in  
Large-Scale Multi-Server Systems**

Committee:

---

Sanjay Shakkottai, Supervisor

---

Sujay Sanghavi, Co-Supervisor

---

François Baccelli

---

Gustavo de Veciana

---

John Hasenbein

---

Sriram Vishwanath

**Resource Allocation in  
Large-Scale Multi-Server Systems**

by

**Sharayu Arun Moharir, B.Tech., M.Tech.**

**DISSERTATION**

Presented to the Faculty of the Graduate School of  
The University of Texas at Austin  
in Partial Fulfillment  
of the Requirements  
for the Degree of

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2014

To my parents.

## Acknowledgments

I would like to thank my advisors Sanjay Shakkottai and Sujay Sanghavi for their invaluable guidance. Sanjay's passion for research, teaching and learning has been a constant source of inspiration. His care, keen interest in my progress and encouragement motivated me to push my limits. In addition to his depth of knowledge, I greatly admire his dedication towards his work and I hope I have internalized at least some of it. Sujay's clarity of thought and his knack for asking the right questions helped me immensely every time I was stuck in my research. I am extremely grateful to him for encouraging me to broaden my horizons by exploring new research areas.

I would like to thank my dissertation committee – Prof. Sriram Vishwanath, Prof. Gustavo de Veciana, Prof. John Hasenbein and Prof. Francois Baccelli for their valuable feedback on this dissertation. I would like to thank Shreeshankar for sharing his insights when I started working on my first research problem and Javad for collaborating with me on the second half of this dissertation.

I would like to thank my friends in Austin – Abhik, Aneesh, Anish, Avhishek, Deepjyoti, Guneet, Ioannis, Kumar, Praneeth, Priti, Priyamvada, Ramya, Sarabjot, Siddhartha, Sindhu, Srinadh, Subhashini and Zrinka who made my stay in Austin enjoyable.

I would not have reached this point in life without the unwavering support of my family. I want to thank my parents for being a constant source of strength, motivation and inspiration, my sister Manjiri for always being there for me and my husband Atit for his love and support.

# **Resource Allocation in Large-Scale Multi-Server Systems**

Publication No. \_\_\_\_\_

Sharayu Arun Moharir, Ph.D.  
The University of Texas at Austin, 2014

Supervisors: Sanjay Shakkottai  
Sujay Sanghavi

The focus of this dissertation is the task of resource allocation in multi-server systems arising from two applications – multi-channel wireless communication networks and large-scale content delivery networks. The unifying theme behind all the problems studied in this dissertation is the large-scale nature of the underlying networks, which necessitate the design of algorithms which are simple/greedy and therefore scalable, and yet, have good performance guarantees.

For the multi-channel multi-hop wireless communication networks we consider, the goal is to design scalable routing and scheduling policies which stabilize the system and perform well from a queue-length and end-to-end delay perspective. We first focus on relay assisted downlink networks where it is well understood that the BackPressure algorithm is stabilizing, but, its delay performance can be poor. We propose an alternative algorithm - an iterative

MaxWeight algorithm and show that it stabilizes the system and outperforms the BackPressure algorithm. Next, we focus on wireless networks which serve mobile users via a wide-area base-station and multiple densely deployed short-range access nodes (e.g., small cells). We show that traditional algorithms that forward each packet at most once, either to a single access node or a mobile user, do not have good delay performance and propose an algorithm (a distributed scheduler - DIST) and show that it can stabilize the system and performs well from a queue-length/delay perspective.

In content delivery networks, each arriving job can only be served by servers storing the requested content piece. Motivated by this, we consider two settings. In the first setting, each job, on arrival, reveals a deadline and a subset of servers that can serve it and the goal is to maximize the fraction of jobs that are served before their deadlines. We propose an online load balancing algorithm which uses correlated randomness and prove its optimality. In the second setting, we study content placement in a content delivery network where a large number of servers, serve a correspondingly large volume of content requests arriving according to an unknown stochastic process. The main takeaway from our results for this setting is that separating the estimation of demands and the subsequent use of the estimations to design optimal content placement policies (learn-and-optimize approach) is suboptimal. In addition, we study two simple adaptive content replication policies and show that they outperform all learning-based static storage policies.



# Table of Contents

|  |            |
|--|------------|
| <b>Acknowledgments</b>   | <b>v</b>   |
| <b>Abstract</b>  | <b>vii</b> |
| <b>List of Tables</b>  | <b>xiv</b> |
| <b>List of Figures</b>   | <b>xv</b>  |
| <b>Chapter 1. Introduction</b>   | <b>1</b>   |
| 1.1 Contribution of the Thesis . . . . .   | 3          |
| 1.2 Structure of the Thesis . . . . .  | 7          |
| <b>Chapter 2. MaxWeight vs. BackPressure: Routing and Scheduling in Multi-Channel Relay Networks</b> | <b>8</b>   |
| 2.1 Introduction . . . . .   | 8          |
| 2.1.1 Related Work . . . . .   | 10         |
| 2.1.2 Contributions . . . . .  | 11         |
| 2.1.2.1 BackPressure Algorithm . . . . .   | 12         |
| 2.1.2.2 SSG BackPressure Algorithm . . . . .   | 12         |
| 2.1.2.3 SSG MaxWeight Algorithm . . . . .  | 12         |
| 2.1.2.4 ILQF MaxWeight and ILQF BackPressure Algorithms . . . . .                                    | 12         |
| 2.2 System Model: 2-Hop Downlink Communication Networks . .  | 13         |
| 2.3 Background: The SSG Scheduling Algorithm . . . . .   | 18         |
| 2.4 Proposed Scheduling and Routing Algorithms for 2-Hop Downlink Networks . . . . .                 | 18         |
| 2.4.1 FD-w/oDL Model . . . . .   | 19         |
| 2.4.1.1 SSG BackPressure for FD-w/oDL . . . . .  | 19         |
| 2.4.1.2 SSG MaxWeight for FD-w/oDL . . . . .   | 19         |

|  |  |           |
|--|--|-----------|
| 2.4.2  | HD-wDL Model . . . . .                                   | 20        |
| 2.4.2.1  | SSG BackPressure for HD-wDL Model . . . . .              | 20        |
| 2.4.2.2  | SSG MaxWeight for HD-wDL Model . . . . .                 | 21        |
| 2.5  | Main Results and Discussion . . . . .                    | 21        |
| 2.5.1  | Stability . . . . .                                      | 22        |
| 2.5.2  | Performance Analysis . . . . .                           | 31        |
| 2.5.2.1  | ILQF BackPressure for FD-w/oDL . . . . .                 | 32        |
| 2.5.2.2  | ILQF MaxWeight for FD-w/oDL . . . . .                    | 33        |
| 2.6  | Simulation Results . . . . .                             | 35        |
| 2.7  | Conclusions . . . . .                                    | 39        |
| <b>Chapter 3. Scheduling in Densified Networks:<br/>Algorithms and Performance</b> |  | <b>40</b> |
| 3.1  | Introduction . . . . .                                   | 40        |
| 3.1.1  | Contributions . . . . .                                  | 42        |
| 3.1.2  | Related Work . . . . .                                   | 43        |
| 3.2  | System Model . . . . .                                   | 45        |
| 3.2.1  | User Mobility . . . . .                                  | 46        |
| 3.2.2  | User-AN Connectivity . . . . .                           | 48        |
| 3.2.2.1  | Unpredictability of user-AN associations . . . . .       | 48        |
| 3.2.2.2  | User-AN associations in consecutive time-slots . . . . . | 49        |
| 3.2.2.3  | Concentration of users around an AN . . . . .            | 50        |
| 3.2.3  | Communication between Access Nodes . . . . .             | 50        |
| 3.2.4  | Interference between Access Nodes . . . . .              | 50        |
| 3.2.5  | Notation . . . . .                                       | 51        |
| 3.3  | Main Results and Discussion . . . . .                    | 53        |
| 3.3.1  | Algorithm: DIST . . . . .                                | 53        |
| 3.3.2  | Stability . . . . .                                      | 57        |
| 3.3.3  | Performance . . . . .                                    | 62        |
| 3.3.3.1  | Single Transmission Algorithms . . . . .                 | 62        |
| 3.3.3.2  | DIST . . . . .   | 63        |
| 3.4  | Proof Outlines . . . . .                                 | 65        |
| 3.4.1  | Stability of DIST (Theorem 9) . . . . .                  | 65        |

|                   |   |            |
|-------------------|---|------------|
| 3.4.2             | Performance Analysis of ST Algorithms (Theorem 10)                          | 67         |
| 3.4.3             | Performance Analysis of DIST (Theorem 11)                                   | 68         |
| 3.5               | Simulation Results  | 70         |
| <b>Chapter 4.</b> | <b>Online Load Balancing Under Graph Constraints</b>                        | <b>76</b>  |
| 4.1               | Introduction  | 76         |
| 4.1.1             | Contributions   | 78         |
| 4.1.2             | Related Work  | 79         |
| 4.1.2.1           | Online Load Balancing with Hard Deadlines                                   | 79         |
| 4.1.2.2           | Online Matching   | 80         |
| 4.2               | System Model  | 81         |
| 4.3               | Main Results and Discussion   | 82         |
| 4.3.1             | Simple Special Case   | 82         |
| 4.3.2             | Upper Bound   | 85         |
| 4.3.3             | Our Algorithm and its Performance   | 86         |
| 4.3.4             | Performance of other Algorithms   | 94         |
| 4.4               | Simulations   | 96         |
| 4.5               | Summary and Discussion  | 96         |
| <b>Chapter 5.</b> | <b>Serving Content with Unknown Demand:<br/>the High-Dimensional Regime</b> | <b>100</b> |
| 5.1               | Introduction  | 100        |
| 5.1.1             | Contributions   | 102        |
| 5.1.2             | Organization and Basic Notations  | 104        |
| 5.2               | Setting and Model   | 104        |
| 5.2.1             | Server and Storage Model  | 104        |
| 5.2.2             | Service Model   | 105        |
| 5.2.3             | Content Request Model   | 106        |
| 5.2.4             | Time Scales of Change in Arrival Process                                    | 106        |
| 5.3               | Main Results and Discussion   | 108        |
| 5.3.1             | Separating Learning from Content Placement                                  | 108        |
| 5.3.2             | Myopic Joint Learning and Placement   | 111        |
| 5.3.3             | Genie-Aided Optimal Storage Policy  | 117        |

|  |  |            |
|--|--|------------|
| 5.4  | Simulation Results . . . . .                             | 123        |
| 5.5  | Related Work . . . . .                                   | 129        |
| 5.6  | Conclusions . . . . .                                    | 131        |
| <b>Chapter 6. On Adaptive Content Replication in Large-Scale Content Delivery Networks</b> |  | <b>133</b> |
| 6.1  | Introduction . . . . .                                   | 133        |
| 6.1.1  | Contributions . . . . .                                  | 136        |
| 6.1.2  | Related Work . . . . .                                   | 138        |
| 6.1.3  | Basic Notation . . . . .                                 | 139        |
| 6.2  | Setting and Model . . . . .                              | 140        |
| 6.2.1  | Catalog Dynamics . . . . .                               | 140        |
| 6.2.2  | Arrivals and Content Requests . . . . .                  | 141        |
| 6.2.3  | Server and Storage . . . . .                             | 142        |
| 6.2.4  | Service and Content Fetching . . . . .                   | 143        |
| 6.3  | Main Results and Discussion . . . . .                    | 143        |
| 6.3.1  | Static Catalog (IRM) . . . . .                           | 143        |
| 6.3.2  | Dynamic Catalog (SNM) . . . . .                          | 149        |
| 6.4  | Simulation Results . . . . .                             | 150        |
| 6.4.1  | Static Catalog (IRM) . . . . .                           | 151        |
| 6.4.2  | Dynamic Catalog (SNM) . . . . .                          | 153        |
| 6.5  | Conclusions . . . . .                                    | 154        |
| <b>Chapter 7. Conclusions</b>  |  | <b>156</b> |
| <b>Appendices</b>  |  | <b>159</b> |
| <b>Appendix A. Proofs from Chapter 2</b>   |  | <b>160</b> |
| A.1  | Large System Stability of Iterative Max Weight . . . . . | 160        |
| A.1.1  | FD-w/oDL . . . . .                                       | 160        |
| A.1.2  | HD-wDL . . . . .   | 177        |
| A.2  | Performance Analysis . . . . .                           | 177        |
| A.2.1  | BackPressure . . . . .                                   | 178        |
| A.2.2  | SSG MaxWeight . . . . .                                  | 178        |

|  |            |
|--|------------|
| A.2.3 ILQF BackPressure . . . . .                                | 184        |
| A.2.4 ILQF MaxWeight . . . . .                                   | 184        |
| A.3 $k$ -hop Stability . . . . .                                 | 185        |
| <b>Appendix B. Proofs from Chapter 3</b>                         | <b>190</b> |
| B.1 Stability . . . . .  | 190        |
| B.2 Performance . . . . .  | 199        |
| <b>Appendix C. Proofs from Chapter 4</b>                         | <b>206</b> |
| C.1 Proof of Proposition 1 . . . . .                             | 206        |
| C.2 Proof of Theorem 12 . . . . .                                | 206        |
| C.3 Proof of Theorem 13 . . . . .                                | 220        |
| C.4 Alternative Algorithms: Proofs of Theorems 14 and 15 . . . . | 223        |
| C.4.1 RANDOMIZED JSQ . . . . .                                   | 223        |
| C.4.2 RANDOMIZED P-JSQ . . . . .                                 | 226        |
| <b>Appendix D. Proofs from Chapter 5</b>                         | <b>229</b> |
| D.1 Proof of Theorem 16 . . . . .                                | 229        |
| D.2 Proof of Theorem 17 . . . . .                                | 238        |
| D.3 Proof of Theorem 18 . . . . .                                | 238        |
| D.4 Proof of Theorem 19 . . . . .                                | 244        |
| D.5 Proof of Theorem 20 . . . . .                                | 245        |
| D.6 Proof of Theorem 21 . . . . .                                | 252        |
| D.7 Proof of Theorem 22 . . . . .                                | 253        |
| <b>Appendix E. Proofs from Chapter 6</b>                         | <b>255</b> |
| E.1 Proof of Theorem 23 . . . . .                                | 255        |
| E.2 Proof of Theorem 24 . . . . .                                | 255        |
| E.3 Proof of Theorem 25 . . . . .                                | 262        |
| E.4 Proof of Theorem 26 . . . . .                                | 265        |
| <b>Bibliography</b>  | <b>276</b> |
| <b>Vita</b>  | <b>292</b> |

## List of Tables

|     |   |     |
|-----|---|-----|
| 5.1 | The performance of the four policies as a function of the system size ( $n$ ) for fixed values of load $\bar{\lambda} = 0.8$ and $\beta = 1.5$ . The values reported are the mean and standard deviation ( $\sigma$ ) of the fraction of jobs served. Both adaptive policies (GENIE and MYOPIC) significantly outperform the two learning-based static storage policies. . . . .        | 126 |
| 5.2 | The performance of the four policies as a function of the Zipf parameter ( $\beta$ ) for fixed values of system size $n = 500$ and load $\bar{\lambda} = 0.9$ . The values reported are the mean and standard deviation ( $\sigma$ ) of the fraction of jobs served. The MYOPIC policy outperforms the two learning-based static storage policies for all values of $\beta$ considered. | 127 |
| 5.3 | The performance of the four policies as a function of the load ( $\bar{\lambda}$ ) for fixed values of system size $n = 500$ and $\beta = 1.2$ . The values reported are the mean and standard deviation ( $\sigma$ ) of the fraction of jobs served. The MYOPIC policy significantly outperforms the two learning-based static storage policies for all loads considered. . . .        | 129 |

## List of Figures

|     |  |    |
|-----|--|----|
| 2.1 | A relay network (Example 1) illustrating that MaxWeight algorithm is not stabilizing. There are four links $(l_1, l_2, l_3, l_4)$ with capacities being $(10, 1, 1, 10)$ packets/slot respectively. The source node is A and the destination is D. . . . .   | 10 |
| 2.2 | An illustrative example of a 2-hop relay network with 2 relays and 3 users. . . . .  | 13 |
| 2.3 | End-to-end delay performance of BackPressure, SSG MaxWeight, ILQF MaxWeight and ILQF BackPressure algorithms for a FD-w/oDL system consisting of 50 users and channels with 2 relays for load = 0.74 and ON-OFF channels with parameters 0.5 and 0.1 for the base-station to relay channels and relay to user channels respectively. . . . . | 36 |
| 2.4 | End-to-end delay performance of SSG MaxWeight, ILQF MaxWeight and ILQF BackPressure algorithms for a FD-w/oDL system consisting of 50 users and channels with 2 relays for load = 0.74 and ON-OFF channels with parameters 0.5 and 0.1 for the base-station to relay channels and relay to user channels respectively. . . . .               | 38 |
| 3.1 | A wireless network with a base-station, densely deployed ANs and mobile users. The users move in and out of the coverage area of the ANs due to mobility, but are always in the coverage area of the base-station. BS/AN image courtesy [41]. . . . .  | 46 |
| 3.2 | Association graph between the ANs and mobile users in the network in Figure 3.1. Each AN is associated with all user that are currently in its coverage range, represented by an edge between the AN and the mobile user. . . . .  | 47 |
| 3.3 | BackPressure v/s DIST: Delay Performance . . . . .   | 71 |
| 3.4 | DIST: Delay Performance for Different Loads . . . . .  | 73 |
| 3.5 | DIST: Delay Performance for Different User Mobility . . . . .  | 74 |
| 3.6 | DIST: Delay Performance for values of the parameter $L$ (number of time-slots the BS waits to let the ANs try and delivery packets to the intended users before directly forwarding it to the user) . . . .  | 75 |

|     |  |     |
|-----|--|-----|
| 4.1 | System Model for Online Load Balancing: An illustration of a system with 3 servers. Job 5 has a server subset $\{x, z\}$ and a deadline of 2 time-slots and the scheduling algorithm needs to decide whether to send the job to server $x$ or $z$ or drop the job. . . . .   | 80  |
| 4.2 | An illustration of our algorithm for the simple case of arrivals only in time 1, with all deadlines being $d_{max} = b = 2$ . Here the server set is $U = \{x, y, z\}$ and the job set is $V = \{1, 2, 3, 4, 5\}$ . In the extended graph, there are 2 copies of each server, so $U_b = \{x1, y1, z1, x2, y2, z2\}$ . Our algorithm picks a random permutation, i.e. ranking, $\pi_b$ of this set $U_b$ and fixes it, as shown on the left. Each vertex in set $V$ is then matched to the highest ranked available vertex in $U_b$ ; this results in the matching on the extended graph, on the left. The figure on the right shows the collapsing back from extended graph to a server allocation; it shows the resulting allocation with max load of 2 on each server. . . . . | 84  |
| 4.3 | INSERT RANKING: Time-line . . . . .  | 87  |
| 4.4 | An illustration of INSERT RANKING. Here the server set is $U = \{x, y, z\}$ and $d_{max} = 2$ . Let all 3 incoming jobs have a deadline of 2 time-slots. The figure on the left is an illustration of INITIALIZE and the figure on the right is an illustration of ONLINE ALLOCATION for time-slot 1. . . . .  | 88  |
| 4.5 | An illustration of SCHEDULE for time-slot 1. Server $x$ serves job 2 since it was matched to $x1$ , server $y$ servers job 3, server $z$ remains idle. . . . .   | 89  |
| 4.6 | An illustration of UPDATE STATE for time-slot 2. . . . .   | 91  |
| 4.7 | INSERT RANKING on the matrix A in Section C.2 . . . . .  | 97  |
| 4.8 | Comparison of INSERT RANKING and RANDOMIZED JSQ . . . . .  | 98  |
| 4.9 | Comparison of INSERT RANKING and RANDOMIZED P-JSQ . . . . .  | 99  |
| 5.1 | Learning-Based Static Storage Policies – <i>The interval <math>T(n)</math> is split into the Learning and Storage phases. The length of time spent in the Learning phase can be chosen optimally using the knowledge of the value of <math>T(n)</math> and the Zipf parameter <math>\beta</math>.</i> . . . . .  | 109 |
| 5.2 | MYOPIC – An adaptive storage policy which changes the content stored on idle servers in a greedy manner to ensure that recently requested content pieces are available on idle servers. . . . .  | 113 |
| 5.3 | GENIE – An adaptive storage policy which has content popularity statistics available for “free”. At time $t$ , if the number of idle servers is $k(t)$ , the $k(t)$ most popular content-types are stored on exactly one idle server each. . . . .   | 119 |



|     |   |     |
|-----|---|-----|
| 5.4 | Plot of the mean values reported in Table 5.1 – performance of the storage policies as a function of system size ( $n$ ) for $\bar{\lambda} = 0.8$ and $\beta = 1.5$ . . . . .  | 128 |
| 5.5 | The mean number of external fetches (content fetched from the back-end server to place on a front-end server) by the two adaptive policies as a function of system size ( $n$ ) for $\bar{\lambda} = 0.9$ and $\beta = 2$ and 3. The first plot shows the performance of both GENIE and MYOPIC. The second plot focuses only on the performance of the MYOPIC storage policy for clarity. . . . . | 132 |
| 6.1 | The SNM model [92] for the dynamics of content catalog and content popularity (arrows show the arrival of new contents to the catalog). . . . .   | 135 |
| 6.2 | An illustration of a CDN with a back-end server and three front-end servers. User requests can be served both by the front-end servers (at low cost) and the back-end server (at high cost). Content can be replicated on the front-end servers by fetching it either from other front-end servers (at low cost) or from the back-end server (at high cost). . . . .                              | 142 |
| 6.3 | LRU-R – An LRU variant that replicates content among several front-end servers. . . . .   | 146 |
| 6.4 | The time interval (denoted by $T(n)$ ) is divided into two phases: Phase 1 – Learning, and Phase 2 – Storage/Optimization; figure adapted from Chapter 5. . . . .   | 147 |
| 6.5 | The cost of content replication policies as a function of system size ( $n$ ) for $\bar{\lambda} = 0.9$ , $\beta = 2$ , $C_f = C_b = C_m = 1$ . . . . .   | 152 |
| 6.6 | The cost of content replication policies as a function of system size ( $n$ ) for $\bar{\lambda} = 0.9$ , $\beta = 2$ , $C_f = 1$ , $C_b = C_m = 10$ . . . . .  | 153 |
| 6.7 | The cost of content replication policies as a function of system size ( $n$ ) for $\bar{\lambda} = 0.9$ , $\beta = 4$ , $C_f = C_b = C_m = 1$ . . . . .   | 154 |
| A.1 | An illustrative example of a 3-hop feed-forward relay network with 2 layers of relays and 3 users. . . . .  | 186 |
| D.1 | Coupled Process . . . . .   | 247 |

# Chapter 1

## Introduction

The focus of this dissertation is the task of resource allocation in multi-server systems arising from two applications – multi-channel wireless communication networks and large-scale content delivery networks. The resource allocation problems we consider are outwardly quite different, as they are motivated from widely differing applications – however they share significant underlying similarities due to the large-scale nature of the underlying networks. For the multi-channel downlink wireless networks we consider, e.g., OFDM-based cellular systems, the number of orthogonal frequency channels as well as the number of users in the system are large. In the large-scale distributed content delivery networks we focus on, both the number of servers and the number of distinct contents in the catalog offered by the network are large. The large-scale of these resource allocation problems necessitates algorithms which are simple/greedy/distributed and thus scalable, yet, have rigorous performance guarantees.

In addition, all resource allocation problems considered in this dissertation have an underlying bipartite graph between servers and jobs which need to be served, where an edge between a job and a server indicates that the server

is equipped to serve the corresponding job. For each application, we model the underlying bipartite graph to capture the characteristics of the system and propose resource allocation algorithms with provable performance guarantees.

For multi-channel wireless communication networks, we adopt a stochastic modeling approach and consider the setting where the graph between jobs and servers is a random process. Stochastic modeling is widely used in the study of point-to-point data transmission in communications networks. In the context of wireless communication networks, an edge between a job and a server (frequency channel) means that the frequency channel can be used to effectively transmit the job (packet) to its destination. To model fading, we consider a time-slotted system where this graph can change across time-slots.

For content delivery networks like Netflix [75] and YouTube [109], a job is a request for a particular content piece (to view/download). Each job has a corresponding server subset equipped to serve that job, which is the set of servers which have that particular content piece stored on them. Therefore, in this context, an edge between a job and a server is equivalent to the server storing the content piece being requested by the job. The classical stochastic modeling approach is not suitable in the context of content delivery networks because stochastic models often presuppose stationary statistics which are hard to come by in a fast evolving setting (e.g. content farms like YouTube) where the statistics of the job arrival process depend on the popularity of various videos, which often changes with time. We consider two settings for resource allocation in content delivery systems. In the first setting, the bipar-

tite graph between jobs and servers is adversarial, and in the second setting, jobs arrive according to an unknown and time-varying stochastic process and the task of resource allocation includes replicating content on servers (i.e., designing the bipartite graph between jobs and servers) in addition to allocating incoming requests to appropriate servers.

Next, we provide a brief summary of our work in each of the problems we consider and then provide a road-map for the rest of the dissertation.

## 1.1 Contribution of the Thesis

In Chapter 2, we study routing and scheduling algorithms for relay-assisted, multi-channel downlink wireless networks (e.g., OFDM-based cellular systems with relays). Over such networks, while it is well understood that the BackPressure algorithm is stabilizing (i.e., queue lengths do not become arbitrarily large), its performance (e.g., delay, buffer usage) can be poor. In this work, we study an alternative – the MaxWeight algorithm – variants of which are known to have good performance in a single-hop setting. In a general relay setting however, MaxWeight is not even stabilizing (and thus can have very poor performance). We study an iterative MaxWeight algorithm for routing and scheduling in downlink multi-channel relay networks. We show that, surprisingly, the iterative MaxWeight algorithm can stabilize the system in several large-scale instantiations of this setting (e.g., general arrivals with full-duplex relays, bounded arrivals with half-duplex relays). Further, using both many-channel large-deviations analysis and simulations, we show that

iterative MaxWeight outperforms the BackPressure algorithm from a queue-length/delay perspective.

With increasing data demand, wireless networks are evolving to a hierarchical architecture where coverage is provided by both wide-area base-stations (BS) and dense deployments of short-range access nodes (AN) (e.g., small cells). The dense scale and mobility of users provide new challenges for scheduling: (i) High flux in mobile-to-AN associations, where mobile nodes quickly change associations with access nodes (time-scale of seconds) due to their small footprint, and (ii) multi-point connectivity, where mobile nodes are simultaneously connected to several access nodes at any time.

In Chapter 3, we study such a densified scenario with multi-channel wireless links (e.g., multi-channel OFDM) between nodes (BS/AN/mobile). We first show that traditional algorithms that forward each packet at most once, either to a single access node or a mobile user, do not have good delay performance. We argue that the fast association dynamics between access nodes and mobile users necessitate a multi-point relaying strategy, where multiple access nodes have duplicate copies the data, and coordinate to deliver data to the mobile user. Surprisingly, despite data replication and no coordination between ANs, we show that our algorithm (a distributed scheduler – DIST) can approximately stabilize the system in large-scale instantiations of this setting, and further, performs well from a queue-length/delay perspective (shown via large deviation bounds).

In Chapter 4, we focus on content delivery networks where each arriving

job may only be served by one of a subset of servers. Such a graph constraint can arise due to several reasons. One is locality of the data needed by a job; for example, in content farms (e.g. in Netflix or YouTube) a video request can only be served by a machine that possesses a copy. Motivated by this, we consider a setting where each job, on arrival, reveals a deadline and a subset of servers that can serve it;. The job needs to be immediately allocated to one of these servers, and cannot be moved thereafter. Our objective is to maximize the fraction of jobs that are served before their deadlines.

For this online load balancing problem, we prove an upper bound of  $1 - 1/e$  on the competitive ratio of non-preemptive online algorithms for systems with a large number of servers. We propose an algorithm - INSERT RANKING - which achieves this upper bound. The algorithm makes decisions in a correlated random way and it is inspired by the work of Karp, Vazirani and Vazirani on online matching for bipartite graphs. We also show that two more natural algorithms, based on independent randomness, are strictly suboptimal, with a competitive ratio of  $1/2$ .

In Chapter 5, we look at content placement in the high-dimensional regime: there are  $n$  servers, and  $O(n)$  distinct types of content. Each server can store and serve  $O(1)$  types at any given time. Demands for these content types arrive, and have to be served in an online fashion; over time, there are a total of  $O(n)$  of these demands. We consider the algorithmic task of content placement: determining which types of content should be on which server at any given time, in the setting where the demand statistics (i.e. the

relative popularity of each type of content) are not known a-priori, but have to be inferred from the very demands we are trying to satisfy. This is the high-dimensional regime because this scaling (everything being  $O(n)$ ) prevents consistent estimation of demand statistics; it models many modern settings where large numbers of users, servers and videos/webpages interact in this way.

We characterize the performance of *any* scheme that separates learning and placement (i.e. which use a portion of the demands to gain some estimate of the demand statistics, and then uses the same for the remaining demands), showing it is order-wise strictly suboptimal. We then study a simple adaptive scheme - which myopically attempts to store the most recently requested content on idle servers - and show it outperforms schemes that separate learning and placement. Our results also generalize to the setting where the demand statistics change with time. Overall, our results demonstrate that separating the estimation of demand, and the subsequent use of the same, is strictly suboptimal.

In Chapter 6, we study content placement in a content delivery network (CDN) where a large number of front-end servers, each with fixed storage and service capacity, serve a correspondingly large volume of content requests. The content placement optimization is driven by online learning, where the requests themselves reveal the relative popularity of the content. Further, content can be transferred in and out of the front-end servers, but by incurring additional cost per transfer. We show that in order to minimize total cost over

time, a simple LRU-like adaptive scheme that myopically attempts to store the most recently requested content on idle servers is asymptotically optimal (asymptotic in scale of system). Further, we show that this adaptive scheme strictly outperforms any learning-based static storage policy (i.e. any policy that uses a portion of the demands to estimate the demand statistics, and then uses this estimate for the optimal static content placement). Thus, our results show that despite the per-transfer cost incurred by adaptive placement, learning-based static storage policies are strictly sub-optimal.

## 1.2 Structure of the Thesis

We present our work in five chapters: Chapter 2 contains our work on routing and scheduling algorithms for relay assisted, multi-channel wireless networks; Chapter 3 presents our work on dense wireless networks with mobile users; Chapter 4 focuses on routing jobs with hard deadlines in content delivery networks in the adversarial setting; our work on content replication in the high-dimensional regime is presented in Chapters 5 and 6. Each chapter motivates the problem and describing the setting before presenting our results and proof outlines. Details of the proofs have been deferred to the Appendices. We conclude with a brief summary of the dissertation in Chapter 7.



## Chapter 2

# MaxWeight vs. BackPressure: Routing and Scheduling in Multi-Channel Relay Networks

### 2.1 Introduction

We consider OFDM (Orthogonal Frequency Division Multiplexing) based multichannel multihop downlink networks consisting of a base-station, relays and users. OFDM based networks are widely being deployed in commercial cellular networks (e.g., LTE [2]); looking forward, it is well recognized that wireless relays are envisioned to be an integral part of the solution for next generation cellular systems (e.g., LTE-Advanced [64]). The setting here – multichannel OFDM wireless networks – is the de-facto standard for 4G cellular communications. These systems have several tens of parallel channels (e.g., WiMax over 20 MHz bandwidth has about 50 channels, with each channel having 25 OFDM sub-carriers grouped together) [14, 15]. A key challenge here is to design good routing and scheduling algorithms that provide good user performance (e.g., small buffer usage, low delay, etc.) <sup>1</sup>

The obvious candidate for scheduling and routing in this scenario is

---

<sup>1</sup>S. Moharir and S. Shakkottai. “MaxWeight vs. BackPressure: Routing and scheduling in multi-channel relay networks.” In proceedings of IEEE INFOCOM, 2013. The coauthors on the paper made equal contributions in obtaining these results.

the BackPressure algorithm [90], which routes and schedules packets based on *differential backlogs* (i.e., queue-length differences from a one-hop downstream node). This algorithm is known to be stabilizing; however, it is known that it can have poor delay performance [107, 18, 88]. An alternative, which simply looks at backlogs and *not* differential backlogs is the MaxWeight algorithm [91]. The MaxWeight algorithm assigns a weight of (queue-length  $\times$  channel-rate), and schedules a collection of links that maximizes the total weight (max-weight independent set). This algorithm is however, *not* stabilizing in general, and thus results in very poor performance. As a simple example, we study the 4-node network in Figure 2.1, where the source node (A) needs to deliver packets at rate 1.5 packets/slot to the destination (D). The only scheduling constraint is that links  $l_1$  and  $l_2$  cannot be activated together. It is clear that with the MaxWeight algorithm, the source node A *always* routes packets along link  $l_1$  (with capacity of 10 packets/slot) and does not utilize the lower path (see figure) due to the scheduling constraint (because the weight of the link  $l_1$  is always 10 times larger than the weight of  $l_2$ ). This results in the buffer at node B becoming arbitrarily large (as the corresponding outgoing link can only support 1 packet/slot). This example seems to indicate that MaxWeight is not a good candidate for relay network scheduling and routing. Surprisingly, we show that the above intuition is *not true* in large-scale downlink networks. We show that for large enough multi-channel downlink relay networks, MaxWeight type algorithms do stabilize the system and have better buffer-usage performance than the BackPressure algorithm. Such smaller buffer usage leads to a

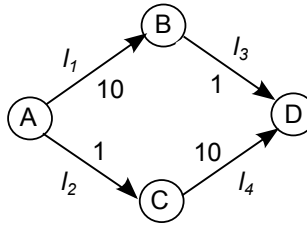


Figure 2.1: A relay network (Example 1) illustrating that MaxWeight algorithm is not stabilizing. There are four links ( $l_1, l_2, l_3, l_4$ ) with capacities being (10, 1, 1, 10) packets/slot respectively. The source node is A and the destination is D.

corresponding smaller packet delay. The intuition that leads to these results is that in networks with a large number of channels (multiple OFDM channels), *(i)* there is sufficient flexibility due to the *degrees of freedom* that the channels provide that can compensate for routing inefficiencies in MaxWeight, and *(ii)* by not considering downlink backlogs, upstream nodes with the MaxWeight algorithm are more aggressive in using good channels to “push” packets closer toward the destination, and thus resulting in better overall performance than BackPressure.

### 2.1.1 Related Work

Performance with MaxWeight and BackPressure algorithms has been studied in many settings over the last decade. With fixed routing (including single-hop flows), delay and buffer-size performance has been studied for mean delay [74, 29] and large buffer asymptotes [108, 96, 83, 87, 97]. Also, from a network stability viewpoint for MaxWeight, work includes [47] where the authors show that the network is stable if the routes are fixed, and nodes are

“decoupled” by means of “measuring” arrival rates [62].

In this work, we focus on properties (stability and queue-length/delay performance) of variants of the BackPressure and MaxWeight algorithms for networks which require dynamic routing.

With dynamic routing and BackPressure like algorithms, modifications have been proposed to queue structures (e.g, shadow queue [18], virtual queues [31], per-hop queues [107]) that empirically result in lower end-to-end delay. Closer to our setting with multiple channels (but only single-hop downlink), large deviation analysis provides buffer-size [14, 15, 16] or delay [85], [48] performance bounds for iterative algorithms.

Our focus here is on downlink multi-hop networks – in this setting, MaxWeight algorithms for *routing* have not been studied (either in single-channel or multi-channel settings) as these algorithms are believed to be not even stabilizing (let alone other performance measures).

### 2.1.2 Contributions

We propose four routing and scheduling algorithms called the Server Side Greedy (SSG) BackPressure algorithm, the SSG MaxWeight algorithm, the Iterative Longest Queue First (ILQF) BackPressure algorithm and the ILQF MaxWeight algorithm in Section 2.4. We show the following:

#### 2.1.2.1 BackPressure Algorithm

- We prove that the BackPressure algorithm does not have good small-queue performance. We show that rate function of the maximum queue length is zero for i.i.d. ON-OFF channels, i.i.d. Bernoulli arrivals, and linear scaling of the number of relays.

#### 2.1.2.2 SSG BackPressure Algorithm

- The algorithm is throughput optimal for the 2-hop networks we consider under general arrival processes, and bounded channel processes.

#### 2.1.2.3 SSG MaxWeight Algorithm

- For 2-hop downlink networks, for arrival rate vectors strictly in the interior the stability region of the system that satisfy some additional constraints, if the system scale is large enough, the algorithm keeps the system stable (see Section 5.3 for specific details).
- For i.i.d. ON-OFF channels, i.i.d. Bernoulli arrivals and linear scaling of the number of relays, we show that the maximum queue length rate function is strictly positive (i.e., exponential decay in queue length tails).

#### 2.1.2.4 ILQF MaxWeight and ILQF BackPressure Algorithms

- For i.i.d. ON-OFF channels, i.i.d. Bernoulli arrivals and linear scaling of the number of relays, we show that the maximum queue length rate function is strictly positive (i.e., exponential decay in queue length tails).

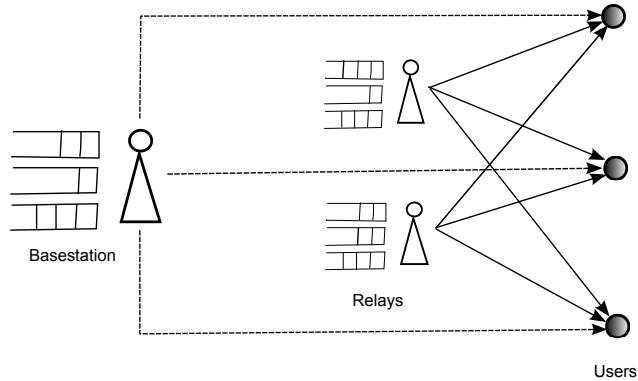


Figure 2.2: An illustrative example of a 2-hop relay network with 2 relays and 3 users.

We compare the lower bounds on the rate functions of the SSG MaxWeight algorithm, the ILQF MaxWeight algorithm and the ILQF BackPressure algorithm and compare their delay performance via simulations. In particular, the bounds for the MaxWeight based algorithms are greater than the bounds for the BackPressure based algorithm and our simulations verify these results.

We finally note that while we have stated and proved the results in the context of 2-hop networks, the results can be easily extended to any  $k$ -hop downlink network (i.e., multiple “layers” of relays). We skip the details to keep notation manageable.

## 2.2 System Model: 2-Hop Downlink Communication Networks

We consider a multiuser, multichannel 2-hop downlink communication system. The system consists of a base-station (BS),  $R(n)$  relays and  $n$  users

and  $n$  channels, the base-station and the relays maintain  $n$  queues each, one for each user in the system as shown in Figure 2.2.

Our results can be generalized to the case where the two quantities (number of users and number of channels) are not equal, but scale linearly with respect to each other. We consider the case when the two are equal to keep the notation simple.

We study a discrete time queuing system. We build on the notation used in [14, 15, 16]. All queue lengths below (i.e., at the BS and relays) are measured at the end of a time-slot  $t$ , and arrivals occur at the beginning of the time-slot.

- $Q_i$  = Queue number  $i$  at the base-station.
- $R_{ri}$  = Queue number  $i$  at relay  $r$ .
- $S_i$  = Channel number  $i$ .
- $Q_i(t)$  = The queue length of user  $i$  at the BS (measured at the end of the time-slot).
- $\mathbf{Q}(t) = \{Q_i(t) : 1 \leq i \leq n\}$ : The vector of queue lengths at the base-station.
- $R_{ri}(t)$  = The queue length of user  $i$  at relay  $r$  (measured at the end of the time-slot).

- $\mathbf{R}(t) = \{R_{ri}(t) : 1 \leq r \leq R(n), 1 \leq i \leq n\}$ : The vector of queue lengths at the relays.
- $A_i(t)$  = The number of arriving packets to  $Q_i$  at the base-station.
- $\mathbf{A}(t) = \{A_i(t) : 1 \leq i \leq n\}$ : The vector of the number of arriving packets at the base-station at the beginning of time-slot  $t$ .
- $A_i^r(t)$  = The number of arriving packets to  $R_{ri}$  (measured at the beginning of the time-slot).
- $X_{i,j}(t)$  = The number of packets in  $Q_i$  that can be transmitted by the BS to user  $i$  on channel  $j$  in time-slot  $t$ .
- $X_{i,j}^{B,r}(t)$  = The number of packets in  $Q_i$  that can be transmitted by the BS to relay  $r$  on channel  $j$  in time-slot  $t$ .
- $X_{i,j}^r(t)$  = The number of packets in  $R_{ri}$  that can be transmitted by the relay  $r$  to user  $i$  on channel  $j$  in time-slot  $t$ .

Note that arrivals to the base-station queues are external and the arrivals to the relay queues are intermediate, i.e., packets sent from the base-station to the relays. We design algorithms that assign channels to the base-station and relay queues in every time-slot, and execute their allocation through the variables  $Y_{i,j}^{B,r}(t)$ ,  $Y_{i,j}^r(t)$  and  $Y_{i,j}(t)$  for  $1 \leq i \leq n, 1 \leq j \leq n$  and  $1 \leq r \leq R(n)$ . These variables are defined as follows:



- $Y_{i,j}(t)$  is 1 if channel  $j$  is scheduled for transmission from  $Q_i$  to user  $i$  in time-slot  $t$  and 0 otherwise.
- $Y_{i,j}^{B,r}(t)$  is 1 if channel  $j$  is scheduled for transmission from  $Q_i$  to  $R_{ri}$  in time-slot  $t$  and 0 otherwise.
- $Y_{i,j}^r(t)$  is 1 if channel  $j$  is scheduled to serve the queue for user  $i$  at relay  $r$  in time-slot  $t$  and 0 otherwise.

The dynamics of the individual queues in the system is described below:

$$\begin{aligned}
Q_i(t) &= \left( Q_i(t-1) + A_i(t) \right. \\
&\quad \left. - \sum_{j=1}^n \sum_{r=1}^{R(n)} X_{i,j}^{B,r}(t) Y_{i,j}^{B,r}(t) - \sum_{j=1}^n X_{i,j}(t) Y_{i,j}(t) \right)^+, \\
R_{ri}(t) &= \left( R_{ri}(t-1) + A_i^r(t) - \sum_{j=1}^n X_{i,j}^r(t) Y_{i,j}^r(t) \right)^+,
\end{aligned}$$

where

$$\begin{aligned}
A_i^r(t) &= \text{the number of packets for user } i \text{ received by relay} \\
&\quad r \text{ at the beginning of time-slot } t.
\end{aligned}$$

We consider the following Interference Models:

1. Full Duplex: In the full duplex model, each relay has two transceivers and therefore, can receive and transmit on the same channels simultaneously.
2. Half Duplex: In the half duplex model, the relays can either receive or transmit in a time-slot.

Using these two interference models, it is possible to construct multiple types of Multihop relay networks. For instance:

1. **Full Duplex without Direct Link (FD-w/oDL)**

In this model, we assume that the relays are full duplex and there is no direct communication link between the base-station and the users. We assume that the interference graph for the relays is a complete graph, i.e., only one of the relays can transmit on a particular channel in a give slot.

2. **Full Duplex with Direct Link (FD-wDL)**

In this model, we assume that the relays are full duplex and there is a direct communication link between the base-station and the users. We assume that the interference graph for the relays is a complete graph.

3. **Half Duplex with Direct Link (HD-wDL)**

In this model, we assume that the relays are half duplex and there is a direct communication link between the base-station and the users. We assume that the interference graph for the relays is a complete graph.

For our results, the interference graph of the relays being a complete graph is the most restrictive condition that can be imposed on interference among the relays. We can show that the same results apply for less restrictive interference constraints. However, we skip the details for brevity. In this chapter, we look at the FD-w/oDL and HD-wDL Models in detail. The results and proofs for FD-w/oDL similarly extend to the FD-wDL Model.

### 2.3 Background: The SSG Scheduling Algorithm

In this section we discuss the Server Side Greedy (SSG) algorithm proposed in [15] which is known to have good delay performance for single hop downlink networks.

The Server Side Greedy (SSG) algorithm was defined in [15] for a single hop downlink system. This algorithm sequentially allocates channels to queues within each time-slot. It first allocates channel  $S_1$  to the maximum weight queue, i.e., the queue with largest  $(Q_i(t)X_{i,1}(t))$ . It updates the queue length based on the number of packets that are drained due to this allocation, and proceeds sequentially to the next channel (and so on). The key point is that even within a time-slot, queue lengths are updated during the allocation process, and future channel allocations within the time-slot take the accumulated queue length drains into account. For a formal definition of the SSG algorithm (and proofs that this has quadratic complexity in  $n$ ), please refer to [15], Definition 3.

### 2.4 Proposed Scheduling and Routing Algorithms for 2-Hop Downlink Networks

The SSG algorithm discussed in Section 2.3 was designed for single hop networks and therefore designed only for scheduling packets. In this section, we build on the SSG algorithm to design scheduling and routing algorithms for multihop downlink networks. We describe the algorithms in the context of 2-hop networks for simplicity, but, they can be extended to  $k$ -hop downlink

networks.

### 2.4.1 FD-w/oDL Model

#### Input:

- The queue lengths  $Q_i(t-1)$  and  $R_{ri}(t-1)$ , for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ .
- The arrival vectors  $A_i(t)$  and  $A_i^r(t)$ , for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ .
- The channel realizations  $X_{i,j}^r(t)$  and  $X_{i,j}^{B,r}(t)$  for  $1 \leq i \leq n$ ,  $1 \leq j \leq n$ ,  $1 \leq r \leq R(n)$ .

#### 2.4.1.1 SSG BackPressure for FD-w/oDL

The allocation for relay queues is carried out first using the SSG rule (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index). The updated relay queue lengths are used for allocation of channels at the BS using the SSG rule with the weight of each link being the backpressure-channel product of that link (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index at each relay).

#### 2.4.1.2 SSG MaxWeight for FD-w/oDL

The allocation for relay queues is carried out first using the SSG rule (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index). The allocation for the BS queues is also done using the SSG rule with the weight of each link being the queue-length-channel product

of that link, breaking ties in a cyclic order as follows. We initialize the priority order of the relays as  $\{1, 2, \dots, R(n)\}$ . In each round of the allocation process, the relay that is allocated that particular channel is then removed from its current position in the priority order and inserted at the last position to get the new priority order.

### 2.4.2 HD-wDL Model

#### Input:

- The queue lengths  $Q_i(t-1)$  and  $R_{ri}(t-1)$ , for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ .
- The arrival vectors  $A_i(t)$  and  $A_i^r(t)$ , for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ .
- The channel realizations  $X_{i,j}^r(t)$ ,  $X_{i,j}^{B,r}(t)$  and  $X_{i,j}(t)$  for  $1 \leq i \leq n$ ,  $1 \leq j \leq n$ ,  $1 \leq r \leq R(n)$ .

#### 2.4.2.1 SSG BackPressure for HD-wDL Model

Let

$$\begin{aligned}\Delta\xi_B(t-1) &= \max_{1 \leq i \leq n, 1 \leq r \leq R(n)} (Q_i(t-1) - R_{ri}(t-1) + A_i(t)), \\ \xi_R(t-1) &= \max_{1 \leq i \leq n, 1 \leq r \leq R(n)} (R_{ri}(t-1) + A_i^r(t)).\end{aligned}$$

If  $\Delta\xi_B(t-1) > \xi_R(t-1)$ , the base-station queues transmit in slot  $t$ , else the relay queues transmit in slot  $t$ . The allocation for relay queues is carried out using the SSG rule (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index). The allocation for the BS queues is

done using the SSG rule with the weight of each link being the backpressure-channel product of that link (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index).

#### 2.4.2.2 SSG MaxWeight for HD-wDL Model

Initialize

$$A_{max} = \max_{1 \leq i \leq n} A_i(0).$$

In each time-slot  $t$ , update

$$A_{max} = \max \left\{ A_{max}, \max_{1 \leq i \leq n} A_i(t) \right\}.$$

Let

$$\begin{aligned} \xi_B(t-1) &= \max_{1 \leq i \leq n} (Q_i(t-1) + A_i(t)), \\ \xi_R(t-1) &= \max_{1 \leq i \leq n, 1 \leq r \leq R(n)} (R_{ri}(t-1) + A_i^r(t)). \end{aligned}$$

If  $\xi_B(t-1) > \xi_R(t-1)$ , the base-station queues transmit in slot  $t$ , else the relay queues transmit in slot  $t$ . The allocation for relay queues is carried out using the SSG rule (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index). The allocation for the BS queues is also done using the SSG rule till all queues have queue length less than  $\xi_B(t-1) - A_{max} - 1$  or we run out of channels to allocate.

## 2.5 Main Results and Discussion

We now state our main results, and discuss their implications.

### 2.5.1 Stability

**Assumption 1:** We use similar Assumptions to [29], [15], described below for completeness.

1. The channel process:

- The channel state process is assumed to have a stationary distribution  $\pi = [\pi]_{i \in I}$ , with  $\pi_i > 0$  for all  $i \in I$  where  $I$  is the collection of possible channel states.
- Denote  $s[m]$  to be the channel state in time-slot  $m$ . We assume that for any  $\epsilon > 0$ , there exists an integer  $M_0 > 0$  such that for all  $M \geq M_0$ , all  $i \in I$ , and all  $k$ , we have

$$E \left[ \left| \pi_i - \frac{1}{M} \sum_{m=k}^{k+M-1} 1_{s[m]=i} \right| \right] < \epsilon.$$

- There exists  $X_{max} > 0$  such that

$$\max_{i,j,t} X_{ij}(t) \leq X_{max}.$$

2. The arrival process:

- The arrival process to each node  $n_i$  in the network is a stationary process with mean  $\lambda_i$ .
- The arrival rates which lie in the interior of the system's throughput region.

- Given any  $\epsilon > 0$ , we assume that there exists an integer  $M_1 > 0$  such that for all  $M \geq M_1$ , and for all  $k, i$ ,

$$E \left[ \left| \lambda_i - \frac{1}{M} \sum_{m=k}^{k+M-1} A_i(m) \right| \right] < \epsilon.$$

- The second moment of the number of arrivals per time-slot is bounded.

For the following theorem, we consider the SSG BackPressure algorithm for any of the models described so far (i.e., FD-w/oDL, FD-wDL, HD-wDL). This theorem continues to hold for any multi-channel network with independent sets based scheduling constraints (in this case, the SSG BackPressure algorithm sequentially allocates max-weight independent sets).

**Theorem 1** (Throughput Optimality of SSG BackPressure). *Under Assumption 1, the SSG BackPressure rule results mean-stable queues, i.e.,*

$$\limsup_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \sqrt{\sum_{i=1}^n Q_i^2(t) + \sum_{i=1}^n \sum_{r=1}^{R(n)} R_{ri}^2(t)} < \infty.$$

As the name suggests, this algorithm takes into account previous channel and user allocations (and the changes in queue lengths due to such allocations) for each successive new channel allocation. The proof of this builds on techniques in [15, 29]. This result shows that the SSG BackPressure algorithm keeps the queues stable, and thus is a candidate for studying other performance measures such as buffer usage or delay. Please refer to [72] for



the proof of this theorem.

**Assumption 2:** (FD-w/oDL: Stability)

• **Assumption 2(a):**

Arrivals and Bounded Channels

- We assume that  $\mathbf{A}(t)$  (the vector of arrivals in a time-slot across users) is an aperiodic, irreducible, finite state Markov chain (independent of the channel process).
- We define  $\lambda = \frac{1}{n} E \left[ \sum_{i=1}^n A_i(0) \right]$ . Then,

$$P \left( \sum_{i=1}^n A_i(t) \geq n(\lambda + \delta) \right) \leq e^{-nk(\delta)},$$

where  $k(\delta) > 0$  is a function of  $\delta$  and independent of  $n$ .

- $A_i(t) \leq k_1 n$  for all  $t$  and  $i$  and some constant  $k_1$ .
- The channel processes are i.i.d. across time-slots.
- $X_{i,j}^{B,r}(t) \leq S_{max} < \infty$ .
- $X_{i,j}^r(t) \leq S_{max} < \infty$ .
- For every  $i, j, r$  and  $t$ ,

$$P(X_{i,j}^r(t) = S_{max}) = q(i, j, r) > 0.$$

• **Assumption 2(b):**

Consider the event  $E$  that there exists a set of channels  $J$  such that

$|J| = nk_2$  for some constant  $k_2 < 1$  and  $X_{i,j}^{B,r} < S_{max}$  for all  $j \in J$  and  $1 \leq r \leq R(n)$ . Then,

$$P(E) = o\left(\frac{1}{n^6}\right).$$

The event  $E$  as described above is equivalent to saying that in a given time-slot, there exists a constant fraction of the channels which cannot be used at  $S_{max}$  by the base-station. If the channels are i.i.d. Bernoulli with parameter  $q$  across relays and time, we have that

$$P(E) = 2^{nH(k_2)}(1-q)^{nk_2R(n)} = o\left(\frac{1}{n^6}\right),$$

where  $H(k_2) = -(k_2 \log(k_2) + (1-k_2) \log(1-k_2))$ . We can show that another sufficient condition is the  $\alpha$  mixing condition defined in [12]. The condition implies that even though the channel variables are not independent, the correlation between them decays over space and time and,  $\alpha$  captures the rate at which correlation decays.

- **Assumption 2(c):**

Let  $I$  be a set of relays such that  $|I| \geq \delta R(n)$ , for some constant  $\delta < 1$ . Consider the event  $G$  that for a channel  $j$  and for every relay  $r \in I$ ,  $X_{i,j}^r(t) < S_{max}$ ,  $\forall i$ . Then,

$$P(G) \leq o\left(\frac{1}{n^4}\right).$$

If the channels are i.i.d. Bernoulli with parameter  $q$  across relays and time, for  $\delta = 0.5$ , we have that

$$P(G) \leq (1-q)^{0.5R(n)}.$$

Therefore, for i.i.d. channels, we need  $R(n) > -\frac{6}{\log(1-q)} \log n$ . The event  $G$  as described above is that given a set of relay which includes  $\delta$  fraction of all the  $R(n)$  relays, none of them can use a channel  $j$  at  $S_{max}$  in a given time-slot.

• **Assumption 2(d):**

Let  $I$  be a set of relay queues such that that  $|I| = k_3 R(n)$  for some constant  $k_3 < 1$  and let  $J$  be a set of channels such that  $|J| = \frac{2k_3 R(n)}{q_{min}}$ , where

$$q_{min} = \min_{r,i,j,t} q(i, j, r, t) > 0.$$

Consider the event  $W$  that for every relay in  $I$  there exist  $k_3 R(n)$  channels in  $J$  such that  $X_{i,j}^r(t) = S_{max}$ . Since  $|J| = \frac{2k_3 R(n)}{q_{min}}$ , for every relay, the expected number of channels in  $J$  such  $X_{i,j}^r(t) = S_{max}$  is at least  $2k_3 R(n)$ . Therefore  $W$  is the event that for all relays, the number of channels which have rate  $S_{max}$  is at least half of its expected value. Then,

$$P(W^c) = o\left(\frac{1}{n^3}\right).$$

If the channels are i.i.d. Bernoulli with parameter  $q$  across relays and time, we have that

$$P(W^c) = k_3 R(n) e^{-\frac{2k_3 R(n)}{q} H(\frac{q}{2}|q)}.$$

This assumption, we can show, is also satisfied by the  $\alpha$  mixing condition defined in [12] and discussed in Assumption 2(b).

*Lemma 1.* Under Assumption 2, if  $\frac{1}{n}E\left[\sum_{i=1}^n A_i(0)\right] = \lambda > S_{max}$ , no scheduling algorithm can stabilize the system.

Therefore,  $\lambda \leq S_{max}$  is a necessary condition for an arrival vector to lie in the stability region of the system.

**Theorem 2.** *Under Assumption 2, for arrival processes with  $\lambda < S_{max}$ , the SSG MaxWeight algorithm stabilizes the FD-w/oDL system, i.e., the markov chain  $\{\mathbf{Q}(t), \mathbf{R}(t), \mathbf{A}(t)\}$  is positive recurrent for  $n > n_0$  where  $n_0$  is a function of  $\lambda$ .*

This is one of the key results of this chapter: For each possible arrival rate vector with mean  $\lambda < S_{max}$  (so that it is strictly within the stability region of the system), if the system scale is large enough, this result shows that the SSG MaxWeight algorithm (that does not use downlink queue lengths) keeps the system stable. As we discussed earlier in Example 1, this is not true in general. The proof leverages the fact that the degrees of freedom resulting from the large number of channels compensates for any possible routing errors due to a lack of knowledge of downlink queues.

This result follows from channel diversity since under Assumption 2(a), the system is stable even if only a finite number of users have non-zero arrival rates.

As mentioned before, this result can be extended to  $k$ -hop networks. Please refer to Appendix A for the details.

**Assumption 3:** (HD-wDL: Stability)

• **Assumption 3(a):**

Arrivals and Bounded Channels

- We assume that the arrival process is stationary, ergodic and i.i.d. across time-slots. We define  $\lambda = \frac{1}{n}E\left[\sum_{i=1}^n A_i(0)\right]$ . Then,

$$P\left(\sum_{i=1}^n A_i(t) = n(\lambda + \delta)\right) \leq e^{-nk(\delta)}.$$

- $A_i(t) \leq k_1 n^\alpha$  for some  $\alpha < 1$ , all  $t$  and  $i$  and some constant  $k_1$ .
- The channel processes are i.i.d. across time-slots.
- $X_{i,j}^{B,r}(t) \leq S_{max} < \infty$ .
- $X_{i,j}^r(t) \leq S_{max} < \infty$ .
- $X_{i,j}(t) \leq S_{max} < \infty$ .
- For every  $i, j, r$  and  $t$ ,

$$P(X_{i,j}(t) = S_{max}) = q(i, j, r) > 0.$$

• **Assumption 3(b):**

Let  $I$  be a set of users such that that  $|I| \geq k_2 n$  for some  $k_2 < 1$ . Consider the event  $G$  that for a channel  $j$  and for every user  $i \in I$ ,  $X_{i,j}(t) < S_{max}$ . Then,

$$P(G) \leq o\left(\frac{1}{n^3}\right).$$

If the channels are i.i.d. Bernoulli with parameter  $q$  across relays and time, we have that

$$P(G) \leq (1 - q)^{k_2 n}.$$

This assumption, we can show, is also satisfied if the  $\alpha$  mixing condition defined in [12] and discussed in Assumption 2(b) holds true for the channel variables.

• **Assumption 3(c):**

Let  $I$  be a set of users such that  $|I| = k_3 n$  and let  $J$  be a set of channels such that  $|J| = \frac{2k_3 n}{q_{min}}$ , where

$$q_{min} = \min_{i,j,t} q(i, j, t) > 0.$$

Consider the event  $W$  that for every user in  $I$  there exist  $k_3 n$  channels in  $J$  such that  $X_{i,j}(t) = S_{max}$ . Then,

$$P(W^c) = o\left(\frac{1}{n^2}\right).$$

If the channels are i.i.d. Bernoulli with parameter  $q$  across relays and time, we have that

$$P(W^c) = nk_3 e^{-\frac{2k_3 n}{q} H(\frac{q}{2}|q)}.$$

This assumption, we can show, is also satisfied if the  $\alpha$  mixing condition defined in [12] and discussed in Assumption 2(b) holds true for the channel variables.

*Lemma 2.* Under Assumption 3, if  $\frac{1}{n}E\left[\sum_{i=1}^n A_i(0)\right] = \lambda > S_{max}$ , no scheduling algorithm can stabilize the system.

Therefore,  $\lambda \leq S_{max}$  is a necessary condition for an arrival vector to lie in the stability region of the system.

**Theorem 3.** *Under Assumption 3, for any arrival process with mean  $\lambda < S_{max}$ , the SSG MaxWeight algorithm stabilizes the HD-wDL system, i.e., the markov chain  $\{\mathbf{Q}(t), \mathbf{R}(t), \mathbf{A}(t)\}$  is positive recurrent for  $n > n_0$  where  $n_0$  is a function of  $\lambda$ .*

Theorems 2 and 3 together form one of the two key messages of this chapter which is that even though MaxWeight type algorithms are not throughput optimal for multihop networks in general, in the setting we consider i.e. large-scale multi-channel downlink networks with relays, they stabilize the system.

The proofs of Theorems 2 and 3 differ from the classical methods of proving stability because of the coupling between the base-station and relay queues. Please refer to Appendix A for the details of the proofs.

We note that the main difference between Assumptions 2 and 3 is that Assumption 2 (FD-w/oDL) is satisfied by all arrival processes such that the mean arrivals for each user is  $\leq kn$  for any constant  $k$  (specifically, any  $k < S_{max}$  works) whereas, Assumption 3 (HD-wDL) only allows arrival process which have mean  $\leq k'n^\alpha$  for any constant  $k'$  and  $\alpha < 1$ . In particular,

this implies that the SSG MaxWeight algorithm with Full Duplex relays can support any point that lie within the interior of the stability region, for  $n$  large enough<sup>2</sup>. This follows because the peak channel rate is  $S_{max}$ ; thus, the maximum rate per user that can be supported by *any* algorithm is no more than  $S_{max}n$ .

On the other-hand, for the Half Duplex system with a direct link, Assumption 3 restricts the per-user arrival process (both mean and peak) to scale no more than  $\leq k'n^\alpha$ . This implies that in this setting, we can provably stabilize systems for which the arrival rates (across users) are more balanced, specifically, no single user can use the entire capacity.

### 2.5.2 Performance Analysis

**Assumption 4:** (FD-w/oDL: Performance Analysis)

- Bernoulli Arrivals and ON-OFF Channels
  - $A_i(t) = \text{Bernoulli}(p)$  i.i.d. across users and time-slots.
  - $X_{i,j}^{B,r}(t) = \text{Bernoulli}(q_2)$  i.i.d. across channels and time-slots.
  - $X_{i,j}^r(t) = \text{Bernoulli}(q_3)$  i.i.d. across channels and time-slots.
- Linearly Scaling Relays

$$R(n) = \tilde{R}n, \tilde{R} > 0.$$

---

<sup>2</sup>Further, we can show that even with a Direct Link between the base-station and the Users, the analogous result goes through.



Our proofs work for any value of  $\tilde{R}$ , however we focus on the more realistic case of  $\tilde{R} < 1$ .

For the case of Bernoulli Arrivals and ON-OFF Channels, in addition to the BackPressure and SSG MaxWeight algorithms we also analyze two other algorithms derived from the Iterated Longest Queue First (ILQF) algorithm introduced in [14] which is known to be buffer-usage rate-function optimal for single hop networks (and thus is a good baseline for comparison).

This algorithm operates iteratively, where, in each iteration the algorithm determines a maximum size matching between the collection of longest queues and ON unallocated channels. After doing so, the queue lengths are updated, and the matching process repeats. The complete description of the algorithm is available in [14], Definition 3. We build on the ILQF algorithm to design scheduling and routing algorithms for multihop downlink networks.

#### **2.5.2.1 ILQF BackPressure for FD-w/oDL**

The allocation for relay queues is carried out first using the ILQF rule (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index). The updated relay queue lengths are used for allocation of channels at the BS using the ILQF rule with the weight of each link being the backpressure of that link (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index at each relay).

### 2.5.2.2 ILQF MaxWeight for FD-w/oDL

The allocation for relay queues is carried out first using the ILQF rule (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index). The allocation for the BS queues is also done using the ILQF rule with the weight of each link being the queue length of that link (tie breaking rule: highest priority is the smallest relay index followed by the smallest user index at each relay).

We now analyze the performance of algorithms of the 4 algorithms for the FD-w/oDL system for the restricted class of arrival and channel processes characterized in Assumption 4. The performance metric we are interested in is the small buffer overflow probability which is the probability that the maximum queue length in the system (both at the base-station and the relays) is greater than a positive integer  $b$ . Formally, for each of these algorithms, we are interested in computing  $c(b)$  where

$$c(b) = \frac{1}{b+1} \min \left\{ \liminf_{n \rightarrow \infty} \frac{-1}{n} \log P \left( \max_{i,r} R_{ri}(0) > b \right), \right. \\ \left. \liminf_{n \rightarrow \infty} \frac{-1}{n} \log P \left( \max_{1 \leq i \leq n} Q_i(0) > b \right) \right\},$$

for any fixed non-negative integer  $b$ .

**Theorem 4.** *Under Assumption 4, for the BackPressure algorithm,*

$$c(b)^{(BP)} = 0.$$

This theorem shows that even though the BackPressure algorithm is throughput optimal, it performs poorly when it comes to keeping the queue lengths small. This empirically holds even in the non-asymptotic region as seen in Figure 2.3.

**Theorem 5.** *Under Assumption 4, for the SSG MaxWeight algorithm, for any  $\epsilon \in (0, 1 - p)$  and*

$$\delta \in \left(0, \frac{q_3(1 - p - \epsilon)}{2 - q_3}\right),$$

$$c(b)^{(SMW)} \geq \min \left( H(p|p + \epsilon), \delta \log \frac{1}{1 - q_3}, \frac{2\delta H(q_3|\frac{q_3}{2})}{q_3} \right).$$

This is the second key result of this chapter. This theorem shows that for the setting that we consider in Assumption 4, the SSG MaxWeight algorithm not only stabilizes the system for  $n$  large enough, but also performs well when it comes to keeping the queue lengths small.

**Theorem 6.** *Under Assumption 4, for the ILQF MaxWeight algorithm,*

$$c(b)^{(IMW)} \geq \min \left( \tilde{R} \log \frac{1}{1 - q_2}, \frac{1}{2} \log \frac{1}{1 - q_3} \right).$$

**Theorem 7.** *Under Assumption 4, for the ILQF BackPressure algorithm,*

$$c^{(IBP)}(b) \geq \min \left( \frac{1}{\lceil \frac{2}{\tilde{R}} \rceil} \log \frac{1}{1 - q_2}, \frac{1}{\lceil \frac{2}{\tilde{R}} \rceil + 1} \log \frac{1}{1 - q_3} \right).$$

Since  $\left\lceil \frac{2}{\tilde{R}} \right\rceil \geq \frac{2}{\tilde{R}} > \frac{1}{\tilde{R}}$  and  $\left\lceil \frac{2}{\tilde{R}} \right\rceil + 1 \geq 2$  for all positive values of  $\tilde{R}$ , we observe from Theorems 6 and 7 that we get better bounds on the rate

function for the ILQF MaxWeight algorithm than the ILQF BackPressure algorithm. The intuition for the improvement is clear: by not considering downlink backlogs, upstream nodes with the ILQF MaxWeight algorithm are more aggressive in using good channels to “push” packets closer toward the destination, and thus we expect, will result in a better performance than ILQF BackPressure. We further observe that the bound for the SSG MaxWeight algorithm in Theorem 5 is independent of  $\tilde{R}$ . Therefore for small enough values of  $\tilde{R}$ , i.e. for a small number of relays, we get better bounds on the performance of the SSG MaxWeight algorithm than the ILQF BackPressure algorithm. However, formally since these are bounds, we compare their relative delay performance through simulations in Section 2.6, which verify the intuition from the bounds.

To prove Theorems 5, 6 and 7 we use technical results on Markov Chain coupling from [16]; however, our algorithm performance analysis substantially differs from [16] as we need to deal with two hops (and can generalize to any finite number of hops), thus introducing coupled queues across hops. This entails a different proof technique.

The good performance of the iterative algorithms comes from the interplay between the large number of channels as well as users.

## 2.6 Simulation Results

We compare the end-to-end delay performance of four algorithms (BackPressure, SSG MaxWeight, ILQF MaxWeight and ILQF BackPressure) for a

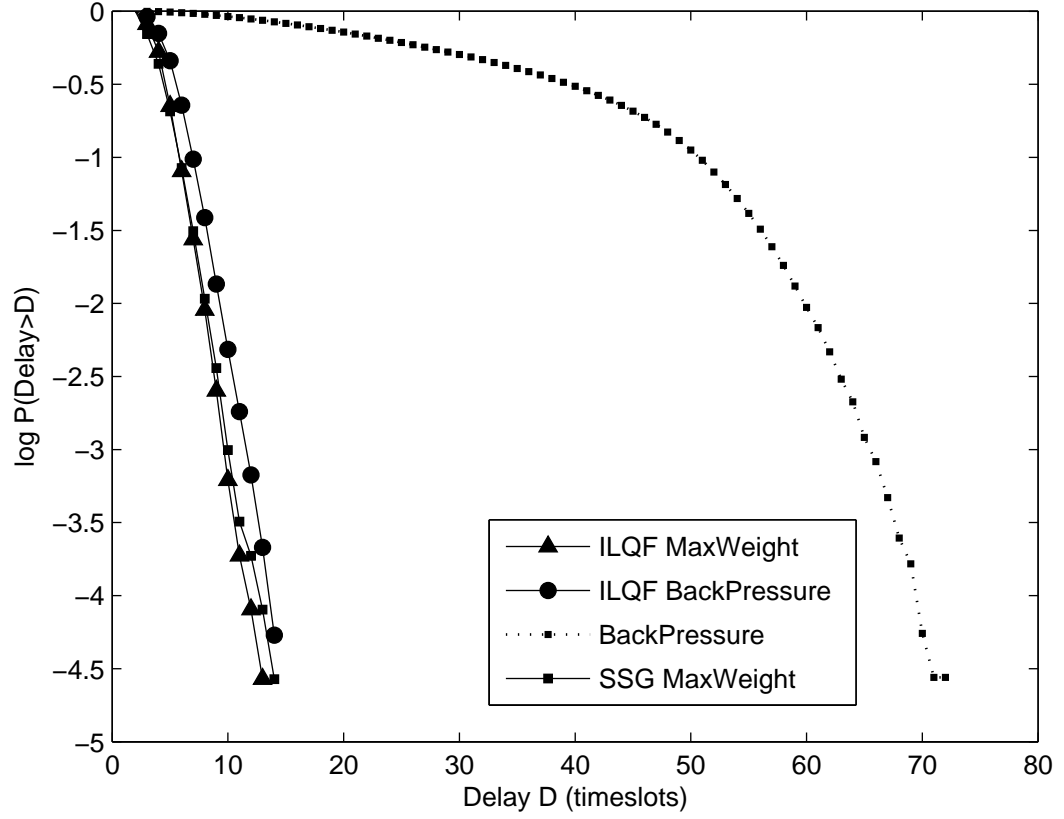


Figure 2.3: End-to-end delay performance of BackPressure, SSG MaxWeight, ILQF MaxWeight and ILQF BackPressure algorithms for a FD-w/oDL system consisting of 50 users and channels with 2 relays for load = 0.74 and ON-OFF channels with parameters 0.5 and 0.1 for the base-station to relay channels and relay to user channels respectively.

FD-w/oDL system. The end-to-end delay of a packet is defined as the number of time-slots it spends in the system before reaching the intended user. This includes the time-slot at the beginning of which the packet arrives at the base-station. We consider end-to-end delay as the metric in the simulations because delay is an important metric for real-time application such as VoIP or video streaming. It is well known that delay is closely related to the queue-length at the base-station and the relays where the packets are temporarily stored on their way to the intended users. Therefore, we expect that algorithms which have good buffer-usage/queue-length performance, also have good end-to-end delay performance.

For this particular experiment, we assume that the system has 50 users, 50 channels and 2 relays. In addition, we assume that  $p = 0.74$ ,  $q_2 = 0.5$ ,  $q_3 = 0.1$ . We ran the system for 10000 time-slots. Figure 2.3 shows the delay performance of all 4 algorithms and Figure 2.4 is the same plot, but zoomed in to get a closer look at the difference in the performance of the three iterative algorithms. We see that the iterative algorithms perform much better than the non-iterative versions. The SSG MaxWeight algorithm seems to be doing better than ILQF BackPressure confirming our intuition that upstream nodes are more aggressive in the SSG MaxWeight algorithm because of the lack of downlink queue length information, leading to better delay performance. This result also validates the difference in the bounds obtained in Theorems 5, 6 and 7.

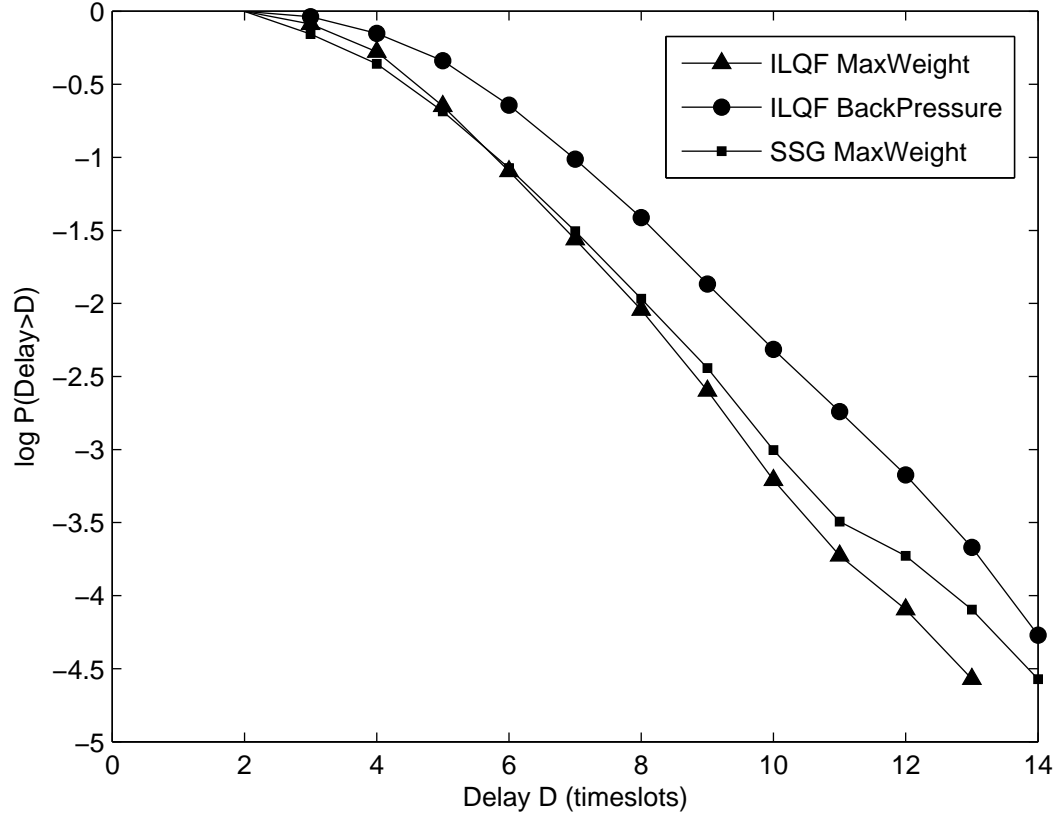


Figure 2.4: End-to-end delay performance of SSG MaxWeight, ILQF MaxWeight and ILQF BackPressure algorithms for a FD-w/oDL system consisting of 50 users and channels with 2 relays for load = 0.74 and ON-OFF channels with parameters 0.5 and 0.1 for the base-station to relay channels and relay to user channels respectively.

## 2.7 Conclusions

We proved that variants of the MaxWeight algorithm are stabilizing for large scale relay networks under appropriate models. We compared the performance of Iterative MaxWeight algorithms and Iterative BackPressure algorithm and found that the Iterative MaxWeight algorithms have better performance. Given that the complexity of these algorithms are not significant (low-degree polynomial, please see [15] for discussion on the complexity of SSG-like algorithms), they can be considered for implementation in practical settings.



## Chapter 3

# Scheduling in Densified Networks: Algorithms and Performance

### 3.1 Introduction

The wireless industry is undergoing a sea change in cellular deployment. From a well-planned macro-cellular setting, the network is evolving to a hierarchical setting with cellular base-stations provide macro coverage (footprint of 1 km or more) and a dense deployment of access nodes (e.g., small cells [86] or femto cells [8, 7]) whose coverage range may be as little as 50 – 100 meters, provides short-range coverage. This combination – macro + dense short-range coverage – popularly referred to as *network densification*, leads to new challenges in network resource allocation.<sup>1</sup>

First, the access nodes’ small footprints imply that mobile nodes associate and disassociate with them at a much higher rate than previously seen. A car moving at just 30 mph results in hand-offs between ANs at the time-scale of seconds. This will likely worsen with emerging technologies for 5G systems

---

<sup>1</sup>S. Moharir, S. Krishnasamy, and S. Shakkottai. “Scheduling in Densified Networks: Algorithms and Performance.” In proceedings of the Annual Conference on Communication, Control and Computing (Allerton), 2014. The coauthors on the paper made equal contributions in obtaining these results.

such as millimeter wave (mmWave) Broadband [79], where the radio propagation environment results in highly non-isotropic and direction-dependent short-range coverage<sup>2</sup>. Thus, to ensure universal coverage, operators have no recourse but to provide a very dense deployment of ANs (especially in locations with high data demand). This leads to second challenge: mobile nodes have the opportunity to associate with several possible ANs at any given time (however, this set changes rapidly over time due to mobility and coverage directionality).

In this chapter, we argue that operating these dense networks in a traditional manner, where mobile nodes associate with one AN at any time, and then hands-off to a new one as the environment/location changes, can be inefficient. Instead, we study an approach where data packets are *replicated* at a collection of ANs whose footprints most-likely cover the mobile node, and these ANs deliver packets to the mobile user by making decisions in a decentralized manner using local information. We communicate directly between the base-station and the mobile node *only as a last resort* when the ANs are unable to reach the mobile node (e.g., due to uncertainty in tracking the mobile node, poor location, poor channel rates due to fading). We propose a formal model to capture this setting and analytically show the performance benefits.

Coordination in wireless communication has been studied in various

---

<sup>2</sup>For instance, in a mmWave Broadband system, the human body completely blocks radio propagation [55, 81]. Thus even slight movement (e.g., rotation of the human with the phone) can completely block the mmWave access node from communicating with a mobile node, thus leading to association changes that can occur within fractions of a second.

contexts like Distributed/Virtual MIMO [78], [76], Network Coding [57], [54] etc. Importantly, these techniques require coordination at the packet or time-slot level. As discussed in [39], backhaul delays could be much larger than the duration of a time-slot; further with densification, heterogeneity in backhaul delays will likely worsen. Thus, the key differentiating aspect from the above literature is that we consider the setting with delayed or sloppy coordination among the various access nodes. In our setting, access nodes do not have current knowledge of nearby nodes' instantaneous states, or indeed, even knowledge of which mobile nodes are connected to them.

### 3.1.1 Contributions

We study scheduling algorithms for networks with a base-station (BS) and multiple densified access nodes (AN) and multiple mobile users. We assume that the ANs are dense enough to support multi-point connectivity, i.e., each user can associate with multiple ANs at any given time. We propose an algorithm (DIST) for scheduling and evaluate its performance as detailed below.

1. **Algorithm DIST:** We propose a distributed algorithm called DIST where the BS and the ANs make their scheduling decisions independently, based only on local channel and queue-length information. Under the DIST algorithm, the BS forwards each packet to an AN that is currently connected to the intended user. If an AN cannot forward a received packet to the corresponding user because the user is no longer

connected to it, unlike traditional algorithms, under the DIST algorithm, the AN forwards copies of packets to multiple ANs around it. In addition, if the ANs fail to deliver a packet to the user within a fixed number of time-slots, it is forwarded directly from the BS to the mobile user.

2. **Stability:** Under general arrival and bounded channel processes, we show that if the system scale is large enough, the DIST algorithm keeps the system stable (i.e., Markovian assumptions imply positive recurrence of the queues).
3. **Performance:** We have two performance results: *(i)* We first show that traditional algorithms like the BackPressure algorithm [90] in which the base-station forwards each packet at most once, either to a single access node or a mobile user, do not have good delay performance for mobile users, i.e., the delay rate functions are zero. *(ii)* For the proposed DIST algorithm, we show that for bounded i.i.d. arrivals and channels, the maximum queue-length rate function is strictly positive and therefore, the queue-length tails decay exponentially. Further, via simulations, we show that the DIST algorithm significantly outperforms the BackPressure algorithm in terms of the delay performance.

### 3.1.2 Related Work

Since the work by Tassiulas and Ephremides [90], there has been great interest in queue-length based scheduling in wireless networks (see [32] for a survey). In the many users/channels context (as in this chapter), there has

been recent activity to characterize stability, queue-length and delay performance, with and without relays (however without user mobility) [14, 15, 16, 85, 71, 46]. A key insight in these works has been the use of iterative allocations, where queues are updated to account for (partial) channel allocations even within a time-slot.

This chapter focuses on the benefits of data replication and multi-point connectivity in a mobile cellular setting (i.e., multiple access nodes maintaining active communications with a mobile user). Such access has had a long history, starting from CDMA soft-handoff (to enable make-before-break voice connections) [98, 104]. More recently, in the setting of COordinated Multi-Point (COMP) [63], there has been much work at the physical layer to develop cooperative communication strategies between a collection of base-stations and a mobile user. This is especially useful in densified settings, with increased opportunities (many base-stations/access points for coordination) and challenges (more complex interference management). These issues have been studied in various ways including simulations [20], field trials [44, 11], and information-theoretic techniques [33] (see [63] for a survey). In this chapter, we focus on network level attributes – queue-lengths and delays – and show that even *local* scheduling algorithms that replicate data can significantly outperform more traditional scheduling algorithms.

Finally, as discussed in the introduction, coordination in wireless networks has a rich history and has been studied in various contexts like Network Coding, Multi-homing, virtual/distributed MIMO etc. See [113] for a discus-

sion of challenges arising at different layers of the network protocol stack as a result of coordination in wireless communication networks. In this chapter, we propose an algorithm which uses only local information, thus obviating the need for coordination between different ANs.

## 3.2 System Model

We consider a two-tiered downlink communication system with a base-station, a large number of ANs and mobile users as shown in Figure 3.1. We study a multi-channel (e.g. OFDM) setting with a large number of orthogonal channels that can be used for communication simultaneously. This multi-channel setting, but without user mobility, was the focus in Chapter 2. However, the fact that users are mobile and that the network is densified implies that the set of ANs that a mobile node is associated with is not time-invariant; further, a classical time-scale decomposition assumption between mobile-AN association and channel scheduling cannot be easily justified.

From a channel (average) rate perspective, our setting is one where the BS-AN, AN-AN and the AN-user links have higher data-rates than the BS-user links. Again, this is a natural setting to consider because the ANs are expected to be mounted in more suitable locations, as well as have superior hardware in terms of the number of antennas, as compared to the mobile users. Moreover, the ANs that a user is associated with are typically much closer to the user than the central BS.

Formally, the system consists of a base-station and  $M(n)$  ANs, where

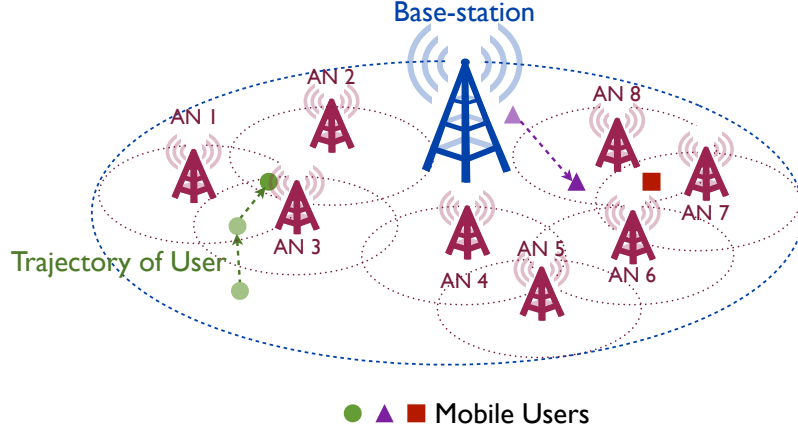


Figure 3.1: A wireless network with a base-station, densely deployed ANs and mobile users. The users move in and out of the coverage area of the ANs due to mobility, but are always in the coverage area of the base-station. BS/AN image courtesy [41].

$n$  is the number of users in the system. We assume that the ANs have two RF chains, one to communicate with the BS and the other to communicate with the users and other ANs. As recommended in [7], the BS-AN communication happens at a different spectrum than the BS-user and AN-user communication. To keep the notation simple, we assume that the number of orthogonal frequency channels for BS-AN communication and AN-user communication are  $n$  each. This setting was also considered in Chapter 2. Our results can easily be extended for other linear scalings.

### 3.2.1 User Mobility

We use a general notion of mobility which allows both fast moving users that move in every time-slot as well as users which move rarely. Formally, we

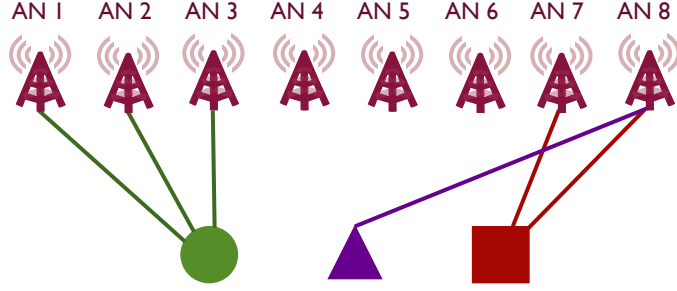


Figure 3.2: Association graph between the ANs and mobile users in the network in Figure 3.1. Each AN is associated with all user that are currently in its coverage range, represented by an edge between the AN and the mobile user.

assume that the probability that a user moves from its current position between two consecutive time-slots is  $\Omega(1/\text{poly}(n))$  (at least of the order of  $1/\text{poly}(n)$ ). This assumption allows the expected time spent at a location to be anything between one and a polynomial function of  $n$ . For example, the probability that a user moves between two consecutive time-slots can be a constant independent of  $n$  as is the case for the Levy-walk process, which is known to be a good model for human mobility in various outdoor settings including college campuses and theme parks [82]. Other popular models, for instance, (discretized versions of) the Random Waypoint Mobility model (RWM) [49] and its variants that have been shown to be more appropriate for user mobility in cellular networks [61], also satisfy this condition.



### 3.2.2 User-AN Connectivity

Since we consider a setting where the ANs are densely deployed, the user is very likely to be connected to multiple ANs. However, we also include the possibility that, in some time-slots, the system fails to obtain the location information of a user. This could happen for various reasons: *(i)* when a user goes out of the coverage area of the ANs, *(ii)* when the user is within the communication range of some ANs, but fails to communicate its position to those ANs, or *(iii)* when a user tracking/position learning algorithm fails. Specifically, we assume that, at the beginning of each time-slot, the location of the user is known with probability at least  $1 - \epsilon(n)$ , i.i.d across users. When the user location is known, it is connected to at least  $C(n)$  ANs. The density of the ANs in the network can be non-uniform, and therefore,  $C(n)$  imposes a lower bound on it. In this chapter we work in the setting where  $C(n)$  is at least  $O(\log n)$ . We also assume that the base-station can always communicate with all users albeit at lower (average) rates than the ANs. Figure 3.2 illustrates the user-AN association graph for the system shown in Figure 3.1.

#### 3.2.2.1 Unpredictability of user-AN associations

Since the users are mobile, the set of ANs a user is connected to can change between two consecutive time-slots. Let  $M_u(t)$  be the set of ANs that user  $u$  is connected to in time-slot  $t$ . We assume that,

- Between two consecutive time-slots, the probability that a connected mobile user,  $u$  moves to a new location such that it is no longer associated

with a previously connected AN,  $m$  is not negligible. Formally, for every  $t$  and  $m \in M_u(t-1)$ ,

$$P(m \notin M_u(t)) \geq \mu_1(n),$$

where  $\mu_1(n) = \Omega(1/\text{poly}(n))$ .

- Further, the motion of the mobile user cannot be predicted with very high accuracy, i.e., for every  $t$  and  $m \notin M_u(t-1)$ ,

$$P(m \in M_u(t)) \leq 1 - \mu_2(n),$$

where  $\mu_2(n) = \Omega(1/\text{poly}(n))$ .

These two conditions are fairly general and are satisfied both by users moving at a very fast time-scale (every time-slot) to users that move rarely ( $\text{poly}(n)$  time-slots in expectation). These conditions are also satisfied by the Levy walk process and the RWM model.

### 3.2.2.2 User-AN associations in consecutive time-slots

We consider the setting where the mobility of users is such that there is some overlap between the ANs a user is connected to in two consecutive time-slots. This imposes a restriction on the maximum velocity of the mobile users. Formally, we assume that, for a user  $u$  connected to the ANs in time-slots  $t$  and  $t+1$ ,  $|M_u(t) \cap M_u(t+1)| = \Omega(\log n)$ .

### 3.2.2.3 Concentration of users around an AN

In dense networks where each AN has a small footprint, it is unlikely that a large number of users will be connected to any one particular AN. Therefore, we can assume that, with high probability, not more than a constant fraction of the total number of users are connected to a particular AN at the same time. Specifically, if  $U_m(t)$  is the set of users connected to AN  $m$  in time-slot  $t$ , then

$$P\left(\max_{1 \leq m \leq M(n)} |U_m(t)| > n^\nu\right) \leq e^{-bn},$$

for a positive constant  $\nu < 1 - \beta$  and a constant  $b > 0$ . This condition is satisfied, for example, if the users are executing a lazy random walk on the network of ANs independent of other users in the system.

### 3.2.3 Communication between Access Nodes

We consider the setting where each AN can communicate with  $O(\log n)$  other ANs located close to it. We assume that the set of ANs that a given AN  $m$  can communicate with is large enough so that even if a mobile user connected to AN  $m$  in time-slot  $t$  moves in the next two time-slots ( $t + 1$  and  $t + 2$ ), AN  $m$  can communicate with at least one AN in  $M_u(t + 2)$ .

### 3.2.4 Interference between Access Nodes

Although the dense deployment of ANs enables multi-point connectivity, it can cause interference at the mobile user due to simultaneous transmissions on the same channel. Let  $I_m$  be the set of ANs which interfere with an

AN  $m$ , i.e., no AN in  $I_m$  can successfully transmit on the same channel as  $m$ . We assume that the interference set for every AN satisfies the following: For all  $m$ ,

$$|I_m| \leq n^\beta,$$

for some constant  $\beta < 1$ . Note that this condition is quite general, and allows for interference sets that grow polynomially in the network size. As a point of reference, spatial stochastic models (where ANs are randomly scattered over the plane), and connectivity as suggested in the Gupta-Kumar model [40] have interference sets that scale only logarithmically in network size, and thus is allowed by our model.

### 3.2.5 Notation

We add to the notation previously used Chapter 2 and [14, 15, 16] to incorporate mobility of users which leads to time-varying user-AN associations. There are  $n$  queues at the base-station and ANs (one per user). Our system evolves in discrete time  $\{t = 0, 1, 2, \dots\}$ , where arrivals happen at the beginning of time-slots, and queues are updated at the end of a time-slot.

- $Q_i, R_{mi}$  = Queue of mobile user  $i$  at the BS and at AN  $m$  respectively.
- $Q_i(t)$  = BS queue-length of mobile user  $i$  at the end of time-slot  $t$ .
- $\mathbf{Q}(t) = \{Q_i(t) : 1 \leq i \leq n\}$ : BS queue-length vector (across all mobile users)

- $A_i(t), A_i^m(t)$  = Number of packet arrivals for mobile user  $i$  at the BS and AN  $m$  respectively at the beginning of time-slot  $t$ .
- $\mathbf{A}(t) = \{A_i(t) : 1 \leq i \leq n\}$ : Arrival vector (across all mobile users).
- $M_i(t)$  = The set of ANs connected to mobile user  $i$  in time-slot  $t$ .
- $C(n) = \min_{i: M_i(t) \neq \emptyset} |M_i(t)|$  is the minimum number of ANs connected to mobile users whose locations are known.
- $U_m(t)$  = The set of mobile users connected to AN  $m$  in time-slot  $t$ .
- $X_{i,j}(t)$  = Channel rate (number of packets) for the  $j$ -th channel from the BS to mobile user  $i$ .
- $X_j^{B,m}(t)$  = Channel rate for the  $j$ -th channel from the BS to AN  $m$ .
- $X_{i,j}^m(t)$  = Channel rate for the  $j$ -th channel from AN  $m$  to mobile user  $i$ .
- $X_j^{l,m}(t)$  = Channel rate for the  $j$ -th channel from AN  $l$  to AN  $m$ .

In each time-slot, channels are allocated to service appropriate queues; this is captured via the decision variables  $Y_{i,j}^{B,m}(t)$ ,  $Y_{i,j}^m(t)$ ,  $Y_{i,j}(t)$  and  $Y_j^{l,m}(t)$  for users  $1 \leq i \leq n$ , channels  $1 \leq j \leq n$  and ANs  $1 \leq m, l \leq M(n)$  (each of the variables takes the value ‘1’ if it corresponds to an allocation, and ‘0’ otherwise).

Finally,  $T_{i,j}(t)$  corresponds to the number of packets transmitted from user  $i$ ’s queue at the base-station on channel  $j$ .

### 3.3 Main Results and Discussion

#### 3.3.1 Algorithm: DIST

In a system implementing this algorithm, the ANs do not cache packets for more than a fixed number of time-slots (say  $L$ ). Any packet which arrives to an AN at the beginning of time-slot  $t$  is deleted by the AN at the end of time-slot  $t + L - 1$ . The base-station stores all packets which have not reached their destination (user).

For each packet  $p \in A_i(t)$  we introduce an indicator variable  $Z_p$  which is 1 if the packet reaches the user  $i$  by the end of time-slot  $t + L$ . The queue-length evolution is now given by:

$$Q_i(t) = \left( Q_i(t-1) + F_i(t) - \sum_{j=1}^n X_{i,j}(t) Y_{i,j}(t) \right)^+,$$

where

$$F_i(t) = A_i(t-L-1) - \left| \sum_{p \in A_i(t-L-1)} Z_p \right|.$$

are the packets which arrived at the BS at the beginning of time-slot  $t-L$ , but, could not be sent to user  $i$  by the end of time-slot  $t-1$ .

We now describe the DIST algorithm (both at the BS and AN).

**Base-Station Algorithm:** The base-station algorithm proceeds in an iterative manner (see [15] for a detailed discussion of iterative algorithms), allocating one channel at a time. Queue-lengths are updated after each round of allocation. Channel  $k$  is allocated in iteration  $k$ .

1: *Forward New Arrivals to ANs*

$$\text{Find } \{i^*, m^*\} \in \underset{1 \leq i \leq n, m \in M_i(t)}{\operatorname{argmax}} A_i^{(k-1)}(t) X_k^{B,m}(t).$$

where  $A_i^{(k-1)}(t)$  is the updated (accounting for packets scheduled for transmission on channels from 1 to  $k-1$ ) number of arrivals to user  $i$  in time-slot  $t$  and  $M_i(t)$  is the set of ANs that user  $i$  is currently connected to. Packets for user  $i^*$  are scheduled for transmission from the base-station to the AN  $m^*$  on channel  $k$ .

2: *Direct Forwarding to Users*

If channel  $k$  is not used by the base-station to forward new arrivals to the ANs, search for the queue index

$$i^* \in \underset{1 \leq i \leq n}{\operatorname{argmax}} Q_i^{(k-1)}(t-1) X_{i,k}(t),$$

breaking ties in the favor of the smaller user index. Allocate channel  $k$  to transmit  $X_{i^*,k}(t)$  from the queue for user  $i^*$  at the base-station directly to user  $i^*$ .

3: *Update Queue-lengths*

Update all queue-lengths before allocating the next channel.

*Remarks.* The salient features of DIST (at the Base-station level) are:

- i. *Step 1 – Local Information + Greedy:* Unlike the BackPressure algorithm, the DIST algorithm does not use differential backlogs to make its routing decisions and therefore does not try to balance the load at the

ANs. Instead, the algorithm tries to push packets to the ANs in a greedy manner whenever it sees high channel rates.

- ii. *Step 2 – Direct Forwarding over Free Channels:* Unused channels (i.e., unused by BS-to-AN transmissions) are used by the base-station to route packets which are queued at the base-station directly to the users. These packets may have previously been successfully received by one or more ANs, but which failed to forward it to the intended user. The base-station transmits these packets directly to the users.

**Access Node Algorithm:** We now describe how each AN carries out the task of channel allocation.

For each AN  $m$ , we define two sets:

- $V_m :=$  the set of ANs that AN  $m$  can communicate with.
- $D_m(t) := \{u : m \in M_u(t-1) \setminus M_u(t)\}$  be the set of users which were connected to AN  $m$  in the previous time-slot, but are not connected to AN  $m$  in this time-slot.

*Remarks.* Before we formally describe the algorithm, the key features of DIST (at the AN level) are:

- i. *Local Information:* Each AN makes its decisions using local queue-length and channel information (channel rates to from  $m$  to ANs in  $V_m$  and users connected to AN  $m$ ).



- ii. *Forwarding Strategy:* For users that are connected to the AN, the AN forwards packets directly to the users. Packets for users that were connected to the AN in the previous time-slot, but are no longer connected to the AN in the current time-slot (users in the set  $D_m(t)$ ), are forwarded to neighboring ANs (ANs in the set  $V_m$ ).
- iii. *Channel Randomization:* Each AN chooses the channel it transmits on uniformly at random from the set of channels which have the highest channel rate. This can lead to collisions, but, since we work in the large scale multi-channel setting, the expected number of collisions are a vanishing fraction of the supportable load.

Formally, each AN implements the following steps:

1: Initialize  $J = \{1, 2, \dots, n\}$  and  $B_u^{l(0)}(t) = A_u^m(t)$  for  $u \in D_m(t)$  and  $l \in V_m$ .

2: *Forward Packets to Connected Users*

If  $\max_{i \in U_m(t)} A_i^{m(k-1)} = 0$ ,  $k = 1$  and goto step 4. Else,

$$\{i^*, j^*\} \in \underset{i \in U_m(t), j \in J}{\operatorname{argmax}} A_i^{m(k-1)} X_{i,j}^m(t),$$

breaking ties uniformly at random. Allocate channel  $j^*$  to serve the queue for user  $i^*$  and update  $J = J \setminus j^*$ .

3:  $A_{i^*}^{m(k)} = (A_{i^*}^{m(k-1)} - X_{i^*,j^*}^m(t))^+$ ,  $k = k + 1$ , and goto Step 2.

#### 4: Forward Packets to Neighboring ANs

$$\{l^*, u^*, j^*\} \in \underset{l \in V_m, u \in D_m(t), j \in J}{\operatorname{argmax}} B_u^{l(k-1)} X_j^{m,l}(t),$$

breaking ties uniformly at random. Allocate channel  $j^*$  to forward packets for user  $u^* \in D_m(t)$  to AN  $l^*$  and update  $J = J \setminus j^*$ .

5:  $B_{u^*}^{l^*(k-1)} = (B_{u^*}^{l^*(k-1)} - X_{j^*}^{m,l^*}(t))^+$ ,  $k = k + 1$ , and goto Step 4.

### 3.3.2 Stability

The DIST algorithm allows the base-station to retransmit packets which have already been received by one or more ANs. Retransmission can lead to the instability of queues in the system, but we show that under some reasonable assumptions on the channel and arrival processes, the DIST algorithm stabilizes the queues in the system. These assumptions are analogous to those in Chapter 2 (see also [29, 15]), with the natural additions to account for user mobility.

*Assumption (a.1).* (Bounded Channel Processes)

- The channel processes are i.i.d. across time-slots (and independent of the arrival process).
- $X_{i,j}^{B,m}(t) \leq C_{max} < \infty$ .
- $X_{i,j}^m(t) \leq C_{max} < \infty$ .
- $X^{m,l} \leq C_{max} < \infty$ .

- $X_{i,j}(t) \leq C_{max}^d < C_{max} < \infty$ .

- For every  $j$  and user  $i$  in time-slot  $t$ ,

$$P(X_{i,j}(t) = C_{max}^d) \geq q_1^{(C_{max}^d)} > 0.$$

- For every  $j$ ,  $t$  and user  $i$  connected to AN  $m$  in time-slot  $t$ ,

$$P(X_{i,j}^m(t) = C_{max}) \geq q_2^{(C_{max})} > 0.$$

- For every  $j$ ,  $t$  and every AN  $l$  which can communicate with AN  $m$ ,

$$P(X_j^{m,l}(t) = C_{max}) \geq q_3^{(C_{max})} > 0.$$

*Assumption (a.2).* (Arrival Process)

- We assume that  $\mathbf{A}(t)$  (arrival vector per time-slot) is an aperiodic, irreducible, finite state Discrete Time Markov Chain.

- $A_i(t) \leq \kappa(n)$  such that  $\kappa(n)\epsilon(n) = o(1)$  and  $\kappa(n)n^\nu n^\beta = o(n^\alpha)$  for some  $\alpha < 1$ .

- We define the load  $\lambda = \frac{1}{n}E\left[\sum_{i=1}^n \left\lceil \frac{A_i(0)}{C_{max}} \right\rceil\right]$ . Then,

$$P\left(\sum_{i=1}^n \left\lceil \frac{A_i(0)}{C_{max}} \right\rceil = n(\lambda + \delta)\right) = o\left(\frac{1}{n}\right),$$

for any  $\delta > 0$ .

Recall that  $\epsilon(n)$  is the probability that a user cannot be located by the ANs in a time-slot; thus requiring the BS to use a (lower rate) channel to directly transmit packets to the mobile. Clearly, as  $\epsilon(n)$  increases,  $\kappa(n)$  has to decrease to maintain stability of the system. The assumption  $\kappa(n)\epsilon(n) = o(1)$  quantitatively captures this effect. For example if  $\epsilon(n) = 1/\sqrt{n}$ , users can have up to  $o(\sqrt{n})$  arrivals in a time-slot.

Recall that w.h.p., the number of users connected to an AN in a time-slot is less than  $n^\nu$  and the size of the interference set for each AN is at most  $n^\beta$ . Therefore,  $\kappa(n)$  has to be small enough to ensure that using  $n$  channels, it is possible for each AN to forward all incoming packets to the corresponding users or other ANs without coordinating with other ANs, yet the number of collisions in each time-slot is a vanishing fraction of the total load on the system.

*Assumption (a.3).* (Base-station to AN Channel Process)

Consider the event  $F_1$  that for channel  $j$ ,  $X_j^{B,m} < C_{max}$  for all ANs user  $i$  is connected to in time-slot  $t$ . This is equivalent to saying that in time-slot  $t$ , channel  $j$  cannot be used at rate  $C_{max}$  by the base-station to forward packets for user  $i$  to the ANs. Then,

$$P(F_1) = o\left(\frac{1}{n^2}\right).$$

*Assumption (a.4).* (AN to Users Channel Process)

For an AN  $m$  and user  $i$  connected to AN  $m$ , consider the event  $F_2$  that there exist at least  $n \frac{q_2^{(C_{max})}}{2}$  channels such that  $X_{i,j}^m(t) = C_{max}$  for each channel.

Then,

$$P(F_2^c) = o\left(\frac{1}{n^3}\right).$$

*Assumption (a.5).* (AN to AN Channel Process)

For an AN  $m$  which can communicate with AN  $l$ , consider the event  $F_3$  that there exist at least  $n \frac{q_3^{(C_{max})}}{2}$  channels such that  $X_j^{m,l}(t) = C_{max}$  for each channel. Then,

$$P(F_3^c) = o\left(\frac{1}{n^3}\right).$$

*Assumption (a.6).* (Base-station to Users Channel Process)

- Let  $I$  be a set of users such that  $|I| \geq kn$ , for some constant  $k < 1$ .

Consider the event  $F_4$  that for a channel  $j$  and for every user  $i \in I$ ,

$X_{i,j}(t) < 1, \forall i$ . Then,

$$P(F_4) = o\left(\frac{1}{n^3}\right).$$

- Let  $I$  be a set of user such that  $|I| = kn$  for some constant  $k < 1$

and let  $J$  be a set of channels such that  $|J| = \frac{2kn}{q_1^{(C_{max}^d)}}$ , where

Consider the event  $F_5$  that for every relay in  $I$  there exist  $kn$  channels

in  $J$  such that  $X_{i,j}(t) = 1$ . Then,

$$P(F_5^c) = o\left(\frac{1}{n^3}\right).$$

For instance, these assumptions (a.3 – a.6) are satisfied by i.i.d. Bernoulli( $q$ ) channels, or more generally, by correlated (across users) channels that have a spatial correlation decay property (modeled via the  $\alpha$ -mixing condition [12]).

**Theorem 8.** *If the load  $\lambda > 1$ , no algorithm can stabilize the queues (i.e., render the queue to be positive recurrent).*

**Theorem 9.** *Under Assumption (a), for a given load  $\lambda < 1$ , there exists  $n_0(\lambda)$  such that for all  $n > n_0(\lambda)$ , the Markov Chain corresponding to the queue-lengths at the base-station and access nodes is positive recurrent.*

Thus, for  $n$  large enough, the DIST algorithm stabilizes the system for all loads  $\lambda < 1$ . Further, this is tight in the sense that beyond  $\lambda = 1$ , we cannot stabilize the queues by any means. This result is interesting because user mobility and collisions at the second hop (AN-user links) lead to retransmissions of those packets by the base-station and yet in the large-scale setting, the DIST algorithm keeps the system stable.

The proof leverages the fact that as the system scale increases, even if a user moves, at least 1 AN that the user is currently connected to has a copy of all the packets which arrived at the base-station less than  $L$  time-slots before the currently time-slot. Therefore, even if the user changes its position, it can receive packets from the ANs it is currently connected to. Moreover, as the number of channels increases, there are sufficient degrees of freedom in the system to ensure that the number of collisions is a vanishing fraction of the supportable load. Therefore, for a given load, as the system scale increases, there is sufficient additional capacity in the system to retransmit packets which are lost due to collisions and directly forward packets from the base-station to those users whose location information is not known. Therefore, we conclude

that, in large scale systems, the benefits of multi-point connectivity can be achieved without the overhead of coordination.

### 3.3.3 Performance

#### 3.3.3.1 Single Transmission Algorithms

We first characterize the performance of a class which we refer to as Single Transmission algorithms. An algorithm belongs to this class if it satisfies the following two conditions:

- i. Each packet is transmitted successfully by the base-station at most once i.e. once the intended receiver (AN/user) of a packet receives it successfully, the base-station deletes that packet from its queue.
- ii. Each AN forwards a received packet only to the corresponding user.

This class of algorithms includes the BackPressure algorithm [90] which is known to be throughput optimal for multihop systems. Iterative versions of the BackPressure algorithm and the MaxWeight algorithm were proposed for multi-channel systems in Chapter 2 and were shown to have good buffer-usage or delay performance for system in which users are not mobile. These algorithms too belong to the ST class of algorithms. The next theorem characterizes the performance of algorithms belonging to the ST class for mobile users.

**Theorem 10.** *For a mobile user in a system implementing an ST algorithm,*

the delay for a packet that is routed to an AN by the base-station is such that

$$d := \limsup_{n \rightarrow \infty} \frac{-1}{n} \log P(\text{Delay} > b) = 0,$$

for any  $b < \infty$ .

We thus conclude that traditional algorithms like BackPressure/MaxWeight [90] do not have good delay performance for mobile users.

### 3.3.3.2 DIST

We study the buffer overflow probability for the largest queue at the base-station:

$$r := \liminf_{n \rightarrow \infty} \frac{1}{b+1} \frac{-1}{n} \log P\left(\max_{1 \leq i \leq n} Q_i(0) > b\right).$$

This value of  $r$  is a bound on the rate of decay of the longest queue (large deviations rate function). Note that the queues at the access nodes delete packets within a small number of time-slots; thus stability or performance of these access node queues is not the focus here.

If an algorithm results in a positive value of  $r$ , then we have that (neglecting constants outside the exponent)

$$P\left(\max_{1 \leq i \leq n} Q_i(0) > b\right) \approx e^{-rn}.$$

Therefore, the probability that the system has any backlogged packets goes to zero very quickly which means that all packets that enter the system as served almost immediately, thus leading to low delay.



We analyze the performance of DIST for a restricted set of arrival and channel processes.

*Assumption (b). (Multi-level Bounded Arrivals and Channels)*

- $A_i(t) = k$  w.p.  $p_k$  for  $0 \leq k \leq K$  and 0 otherwise
- $X_{i,j}^{B,m}(t) = c$  w.p.  $q_1^{(c)}$  for  $0 \leq c \leq C_{max}$  and 0 otherwise.
- $X_{i,j}^m(t) = c$  w.p.  $q_2^{(c)}$  for  $0 \leq c \leq C_{max}$  if user  $i$  is connected to AN  $m$  and 0 otherwise.
- $X_j^{m,l}(t) = c$  w.p.  $q_3^{(c)}$  for  $0 \leq c \leq C_{max}$  if AM  $m$  can communicate with AN  $l$  and 0 otherwise.
- $X_{i,j}(t) = 1$  w.p.  $q_4$  and 0 otherwise.
- $\epsilon(n) = o(1)$ .

The arrival and channel processes are i.i.d. across users, ANs and time-slots. In addition we assume that  $C(n) \geq 2 \log n$ .

**Theorem 11.** *Under Assumption (b), for the DIST algorithm, for any integer  $b \geq 0$ ,*

$$r = \liminf_{n \rightarrow \infty} \frac{-1}{n} \log P \left( \max_{1 \leq i \leq n} Q_i(0) > b \right) > 0.$$

From this theorem we conclude that under Assumption (b), using multi-point connectivity, good buffer-usage performance can be achieved without the overhead of multi-point coordination.

Like the proof of Theorem 9, this proof too leverages the fact that as the system scale increases, multi-point connectivity and the large number of channels ensure that the number of collisions is small and direct retransmission of packets from the base-station to users ensures that no packets stays in the system for too long.

### 3.4 Proof Outlines

In this section, we provide proof outlines for some of the key theorems.

#### 3.4.1 Stability of DIST (Theorem 9)

Stability of multihop systems has been studied in literature in numerous settings, Chapter 2 being closest to the setting in the chapter. In Chapter 2, stability of a static multihop system (no user mobility) for an iterative version of the MaxWeight algorithm was proved in a sequential manner by first showing the stability of base-station queues followed by showing that the relay queues are also stable. The reason why such a decoupling is possible in Chapter 2 is that the MaxWeight algorithm is an ST algorithm and therefore, once a packet is forwarded by the base-station to a relay/user, it is deleted from the queue at the base-station. The queue process at the base-station is therefore independent of the packet transmissions at the second hop (relay-user links). However, for the DIST algorithm, every packet in the system which has not reached its final destination (user) is queued the base-station even if it has been forwarded to the ANs. This couples the queue processes at the base-station

with the channel allocation at the second hop (AN-user links).

Therefore, unlike Chapter 2, where stability was proved in a sequential manner, we have to analyze the entire system at once which requires a different proof structure. Moreover in our setting, since all packets which have not reached their destination (user) are queued at the base-station, it suffices to show that the base-station queues are stable in order to show stability of the system.

Apart from this key difference, the analysis of DIST has three other new aspects.

1. *Dealing with missing user location information:* Unlike settings considered previously, we deal with users whose location is sometimes unknown. We show that there is sufficient unused capacity for DIST to directly forward packets to such users (see also (3) below).
2. *Decentralized nature of DIST:* The ANs forward packets received to connected users and scheduling decisions are made in a distributed manner. This can lead to two bad events: (i) there are packets which no AN forwards to a user, and (ii) due to collisions, packets are not received successfully by the users. We show that for the DIST algorithm, the number of such bad events in the second hop (AN-user links) is  $o(n)$  with probability  $\geq 1 - o(e^{-n})$ .
3. *Splitting packets at the BS into new and old packets:* The base-station forwards new arrivals to the ANs and old packets (packets that arrived

more than  $L$  time-slots before the current time-slot) directly to the users. We show that all new arrivals for users whose location is known are forwarded to the ANs by the BS in a given time-slot with probability  $(\geq 1 - o(1/n))$ . We then show that there is sufficient additional capacity in the system (channels unused by the BS-AN links) to ensure that all packets that arrived  $L$  slots before the current time-slot  $t$ , and which could not be forwarded by the ANs either due to collisions due to the decentralized nature of DIST or because the location of those users was not known can be sent directly from the base-station to the users in time-slot  $t$ .

Using these properties of the DIST algorithm, we show that on average, the base-station queues can serve more packets than they receive in a time-slot (accounting for both new arrivals and old packets that re-enter the base-station queues because the ANs fail to forward them to the users). We then use the standard Foster's Lyapunov technique for Markov Chains (with a quadratic Lyapunov function) to show stability. In other words, for a given load  $\lambda$ , there exists a constant  $n_0$  such that the DIST algorithm ensures that the base-station queues in a system with  $n > n_0$  channels are positive recurrent.

### 3.4.2 Performance Analysis of ST Algorithms (Theorem 10)

1. We consider a packet  $p$  for a mobile user  $u$  which is sent to AN  $m$  by the base-station in time-slot  $t$ . Let  $F$  be the event that the user never connects to  $m$  in time-slots  $t + 1$  to  $t + b$ . By the assumptions made in

Section 3.2,

$$P(F) \geq \min\{\mu_1, \mu_2\} \cdot \mu_2^{b-1}$$

Since  $\mu_1, \mu_2 = \Omega(1/\text{poly}(n))$ , we have that  $P(F) = \Omega(1/\text{poly}(n))$ .

2. Conditioned on  $F$ , the packet cannot reach the user  $u$  before the end of time-slot  $t + b$ . Therefore we conclude that,

$$d = \limsup_{n \rightarrow \infty} \left( -\frac{1}{n} \log P(\text{delay} > b) \right) = 0.$$

### 3.4.3 Performance Analysis of DIST (Theorem 11)

We use Markov Chain coupling results in [16] to prove this theorem. Unlike in [16] where the coupling results were introduced, or in Chapter 2 where multihop static networks were studied using similar coupling arguments, the analysis of the DIST algorithm for the setting in this chapter is more challenging because of user mobility, missing user location information, collisions at the second hop (AN-user links) and most importantly, due to coupling between the base-station queues and transmissions on the AN-user links as discussed in the proof outline for Theorem 9. Instead of sequentially looking at base-station queues followed by relay queues as in Chapter 2, we characterize the buffer-usage at the base-station queues since all packets which have not reached their final destination are queued at the base-station.

Our key step is the construction of a new random variable  $Y^{(n)}(t)$  which dominates the maximum queue-length process at the base-station in our system.  $Y^{(n)}(t)$  is a Markov Chain which increases by at most a constant with

a very small probability ( $e^{-rn}$ , for some constant  $r > 0$ ) and decreases by ‘1’ with constant probability. This construction enables us to use coupling results from [16] to lead to the desired result.

The first step in the construction is to show that for the base-station queues, the probability that the maximum queue-length increases in a slot is small ( $\leq e^{-nr}$ ). We do this via the following steps:

1. *Number of Collisions is Small:* From the proof of Theorem 2 (stability of DIST), we know that the number of bad events (collisions and packets not being forwarded by ANs) in the second hop (AN-user links) is  $o(n)$  with probability  $\geq 1 - o(e^{-n})$ .
2. *Most new arrivals reach users in  $L$  time-slots:* We show that all new arrivals to the base-station for users whose location is known are forwarded to the ANs in a given time-slot with high probability ( $\geq 1 - e^{-nr}$ ) and by (1), we know that all but  $o(n)$  of them are forwarded to the users or ANs connected to that user in the next two time-slots.
3. *Direct Transmission of Old Packets:* We show that there is sufficient additional capacity in the system (channels not used by the BS-AN links for new packets) so that all packets which could not be forwarded by the ANs to the users by  $L$  time-slots after their arrival into the system can be sent directly from the base-station to the users in time-slot  $t$  with high probability ( $\geq 1 - e^{-nr}$ ).

The above steps ensure that the maximum queue-length does not increase (with exponentially high probability). When combined with long-term stability arguments as in Lemma 8, [15], it can be shown that within a finite number of time-steps, the maximum BS queue-length (in the dominating system) decreases by a fixed amount with constant probability. Finally, by explicitly analyzing the dominating system, we obtain the desired bounds on the decay rate of the maximum base-station queue-length.

### 3.5 Simulation Results

We compare the delay performances of the DIST algorithm and the BackPressure algorithm and also study the effect of various system parameters on the delay performance of DIST via simulations. We choose the BackPressure algorithm as a benchmark because it is known to be throughput-optimal for multihop networks.

We consider a system consisting of a base-station, 50 users, 50 channels and 50 ANs and run it for  $10^4$  time-slots. We assume that the ANs lie on a line. The mobility of users is a lazy random walk on this line. Unless specified, each user is assumed to be connected to the three nearest ANs. The BS-AN channels and the AN-User channels take the value 0, 1 and 2 with probabilities 0.2, 0.3 and 0.5 resp and the BS-User channels take the value 1 and with probabilities 0.8 and 0 otherwise, i.i.d. across users, ANs and time-slots. We assume that  $\epsilon(n) = 0.01$ . The following plots show delays for arrivals across time-slots and users (the arrival process is symmetric).

Figure 3.3 compares the performance of DIST with  $L = 5$  and the BackPressure algorithm. In this plot, the parameter Mobility is defined to be the probability that a user moves between two consecutive time-slots. The load on the system in this plot is 0.7. There is a significant difference in the performance of the two algorithms.

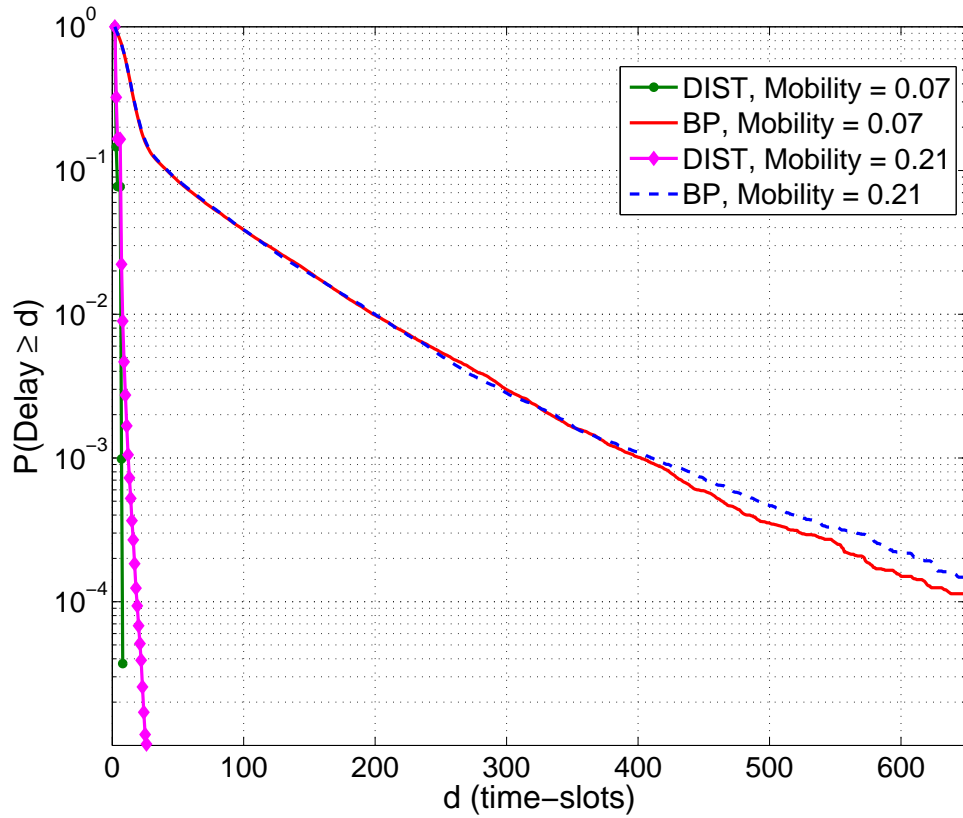


Figure 3.3: BackPressure v/s DIST: Delay Performance

Figure 3.4 summarizes the performance of DIST with  $L = 5$  for three



different loads. Since  $L = 5$ , if a packet does not reach its destination (intended user) in 5 time-slots, the base-station tries to forward it to the mobile user directly. Such packets reach the user with a delay of at least  $L + 1$ . From Figure 3.4, we see that most packets reach their destination in  $L + 1$  ( $=6$ ) time-slots. As expected, packet delays increase with load, but, compared to the performance of the BackPressure algorithm in 3.3, the delay performance of DIST is significantly better even for higher loads.

Figure 3.5 compares the performance of DIST with  $L = 5$  for different values Mobility (the probability that a user moves between two consecutive time-slots) and load = 0.7. The delay performance worsens as the mobility of users in the system increases, but, remains comparable even when the probability a user moves between two consecutive time-slots triples. We conclude that the delay performance of the DIST algorithm is quite robust to user mobility.

Figure 3.6 compares the performance of DIST for load = 0.7 and Mobility = 0.1 for different values of the parameter  $L$  which is the number of time-slots the BS waits to let the ANs try and delivery packets to the intended users. If a packet does not reach the intended user via an AN within  $L$  time-slots after its arrival, the BS directly sends it to the user. As can be seen in Figure 3.6, most of the packets reach their final destination within  $L + 1$  slots after their arrival.

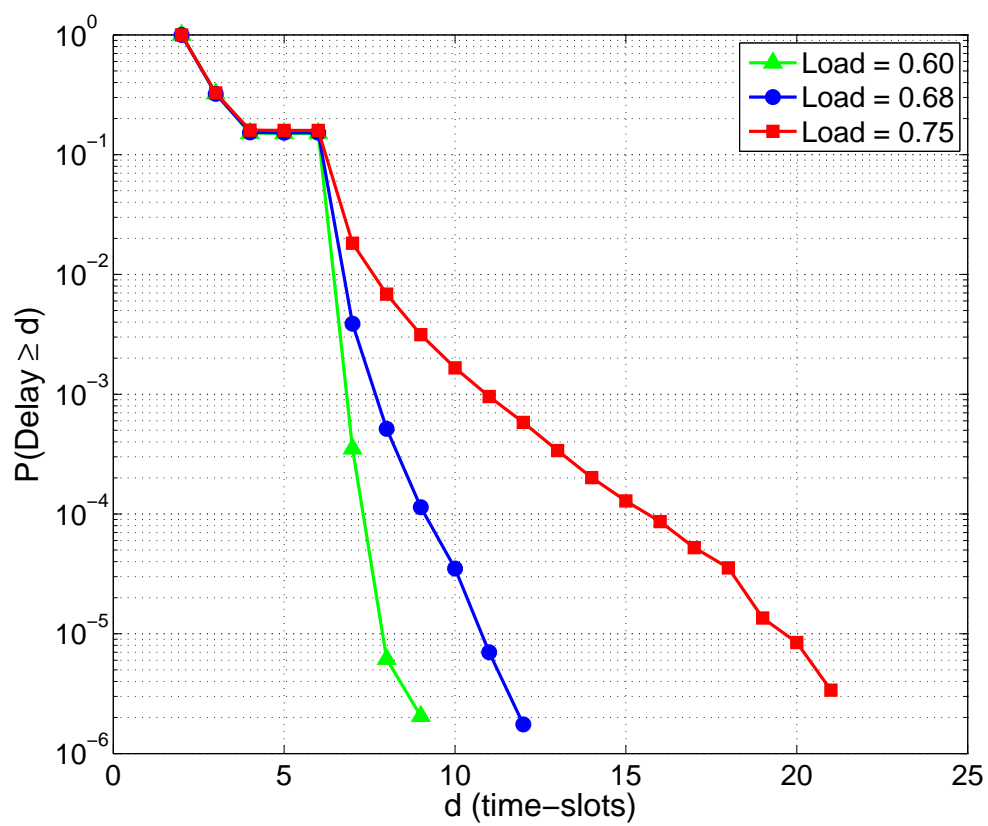


Figure 3.4: DIST: Delay Performance for Different Loads

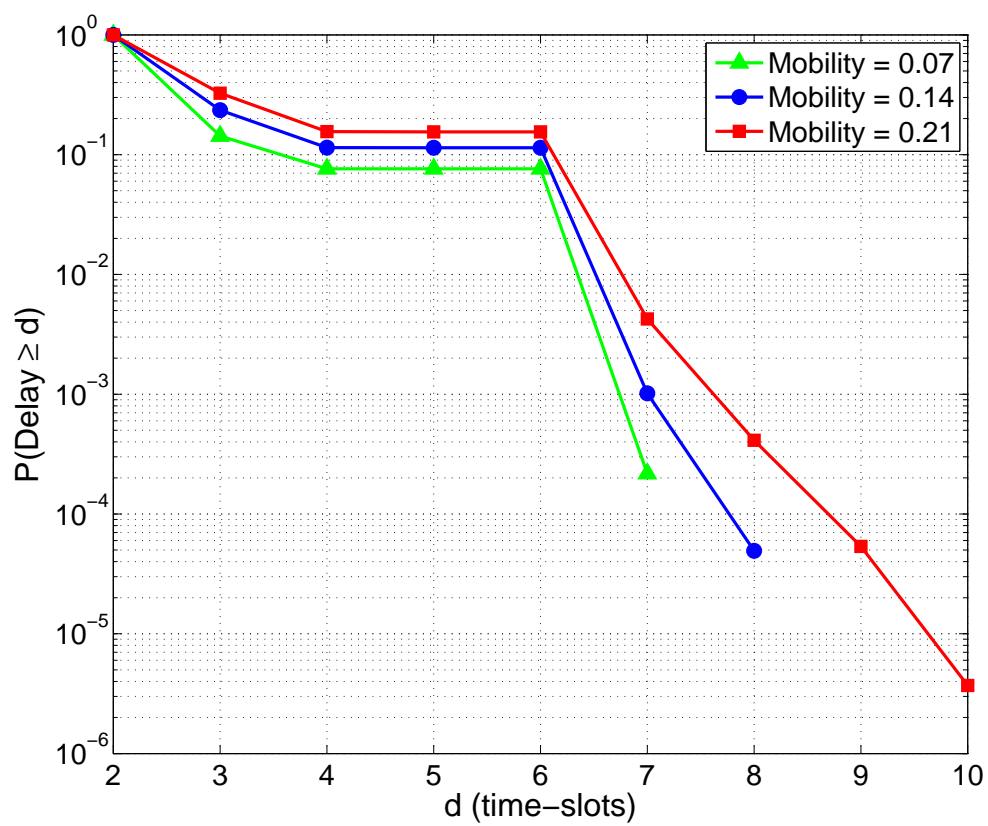


Figure 3.5: DIST: Delay Performance for Different User Mobility

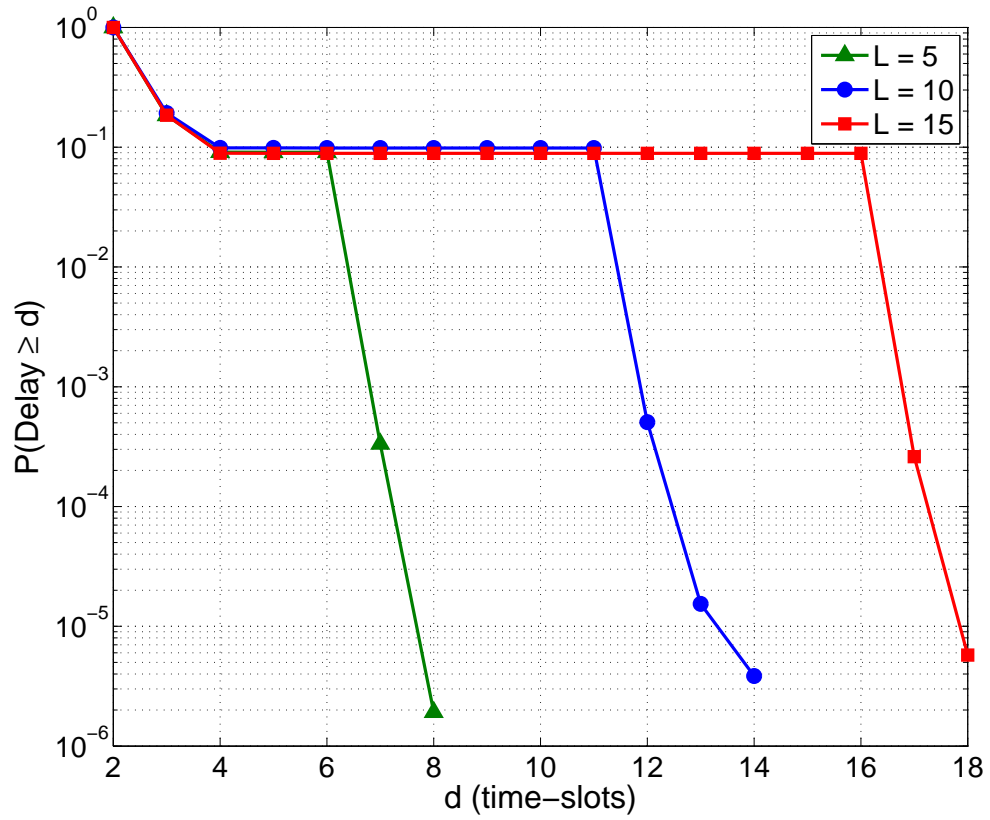


Figure 3.6: DIST: Delay Performance for values of the parameter  $L$  (number of time-slots the BS waits to let the ANs try and delivery packets to the intended users before directly forwarding it to the user)

## Chapter 4

# Online Load Balancing Under Graph Constraints

### 4.1 Introduction

We look at the problem of load-balancing among servers in the setting where each job can only be served by a restricted subset of the servers. Such a constraint arises in several settings, including in content farms and spatially distributed cloud servers. In content farms like Netflix [75] and YouTube [109], videos are replicated only among a subset of the servers; thus a request for a specific video can be served only by the corresponding subset. As another example, web portals that provide a collection of services (e.g., Google providing Email, Maps, Video, Storage and News services) might not have all services replicated on every server. This naturally leads to an association between each request and a subset of servers, based on the request type. Further, the content requests typically have a short fuse, and requests need to be served in (near) real-time.<sup>1</sup>

---

<sup>1</sup>S. Moharir, S. Sanghavi, and S. Shakkottai. “Online load balancing under graph constraints.” In proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems. ACM, 2013. The coauthors on the paper made equal contributions in obtaining these results.

The above discussion motivates the problem in this chapter: We consider online load balancing of jobs with deadlines on non-identical servers (i.e., jobs have “hard” server preferences). In other words, each job has a deadline and can be served by only a (job-specific) subset of the servers, and these preferences are revealed only when the job enters the multi-server queuing system.

Our objective is to design online algorithms which maximize the fraction of jobs that are served before their deadlines. In this work, the performance of an online algorithm is compared with the performance of a non-causal scheduler that has information of all future arrivals. We assume that the online algorithm has no knowledge of the statistics of the job arrival process. We characterize the performance of an online algorithm by its competitive ratio which is the ratio of the expected number of jobs served by the online algorithm to the number of jobs served by the optimal offline algorithm, minimized over all input sequences.

We visualize the system as a bipartite graph between servers and jobs, where an edge between a job and a server indicates that that particular job can be served by that particular server. The task of allocating jobs to servers is equivalent to the task of finding a generalized matching in the bipartite graph where a generalized matching is a matching where one server could be matched to multiple jobs but one job is matched to at most one server.

#### 4.1.1 Contributions

1. **Upper Bound:** We show an upper (i.e. outer) bound of  $1 - 1/e$  on the competitive ratio of *any* randomized load balancing algorithm (Theorem 12).
2. **Algorithm and its Performance:** Our main contribution is the INSERT RANKING algorithm. If multiple servers can serve a job before its deadline elapses, INSERT RANKING breaks ties in a randomized manner and the tie breaking policy is *correlated* across jobs arriving in a time-slot and also across multiple time-slots. We show a lower bound (i.e. achievable) of  $1 - 1/e$  on the competitive ratio of INSERT RANKING (Theorem 13). This proves the optimality of INSERT RANKING.

3. **Other Algorithms:**

We analyze the performance of two intuitive randomized algorithms which do not use correlation in tie-breaking. One is a join the shortest queue algorithm which breaks ties uniformly at random *independent* of past choices, and the second algorithm is biased towards joining shorter queues and breaks ties in a random manner, independent of past choices. We show an upper (i.e. outer) bound of  $1/2$  on the competitive ratio of both algorithms and show that INSERT RANKING strictly outperforms them, highlighting the importance of correlated randomization (Theorem 14 and Theorem 15).

## 4.1.2 Related Work

### 4.1.2.1 Online Load Balancing with Hard Deadlines

There is a vast amount of literature on load balancing. We focus on load balancing in the adversarial setting and similar to our setting of jobs with hard deadlines.

Online admission control in the hard deadline model when there is a single server that services a sequence of jobs in a non-preemptive manner has been studied in [36]. Load balancing jobs with hard deadlines for systems with  $m$  identical machines has been studied in [26] for jobs with identical processing time. In [26], the goal is to minimize the total number of jobs dropped from the system. The key results are an optimal  $3/2$  competitive online algorithm for  $m = 2$  and lower bounds on deterministic algorithms for the general case. The case of  $m = 2$  has also been studied in [37]. The scheduling problem for jobs with equal processing times has been studied in [21] where the goal is to schedule jobs on a single-processor in a non-preemptive manner to maximize the number of completed jobs. Load balancing for batch arrivals of jobs with equal processing times has been considered in [10]. The competitiveness of online deadline scheduling problems where jobs are non-preemptive and the goal is to maximize the sum of the length of jobs completed before their deadlines has been analyzed in [60]. Online preemptive scheduling problem for jobs with deadlines has been studied in [19]. Closer to our setting of trying to maximize the number of jobs completed before their deadlines where all jobs need a service of one time-slot, [28] looks at load balancing



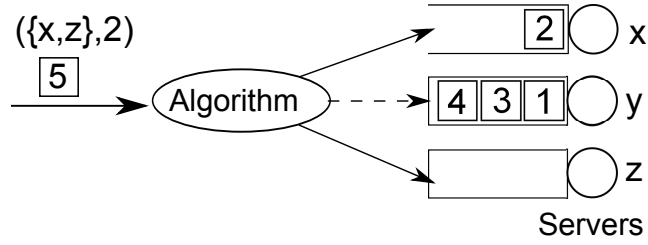


Figure 4.1: System Model for Online Load Balancing: An illustration of a system with 3 servers. Job 5 has a server subset  $\{x, z\}$  and a deadline of 2 time-slots and the scheduling algorithm needs to decide whether to send the job to server  $x$  or  $z$  or drop the job.

these jobs on  $m$  *identical* machines and show a lower bound of  $e/(e-1)$  for large values of  $m$ . Scheduling jobs with deadlines for wireless networks has been studied in [42, 45, 84, 80, 27, 73].

#### 4.1.2.2 Online Matching

The problem of online matching in bipartite graphs has been studied in [53], where the authors introduced the idea of using correlated randomness for online matching. Various extensions to weighted graphs and the application of online weighted bipartite matching to ad-words has been studied in [67, 3, 35]. The problem of Online  $b$ -Matching has been studied in [50]. Closer to our setting, load balancing for bipartite graphs has been studied in [69] in the case where there are no departures from the system and no deadlines.

## 4.2 System Model

We consider a multi-server discrete time queueing system. We assume that jobs come into the system at the beginning of each slot. If there are multiple arrivals in a slot, they are ordered. Each job can only be served by a subset of the servers; *this subset is revealed only when a job arrives*. Each job takes 1 time slot, on any of the servers that can serve it. Furthermore, every job  $p$  has a *deadline*  $d_p$  associated with it; this is the maximum number of slots it can wait, after arrival. There exists a finite maximum  $d_{max} < \infty$  such that  $d_p \leq d_{max}$  for all  $p$ . See Figure 4.1 for an illustration of the system model.

Our task is to allocate jobs to servers so as to maximize the number served by their deadline. Any job, once allocated, cannot be revoked / moved to another choice. Thus, a job  $p$  will not be served if, at the time of its arrival, all the servers it can be allocated to already have more than  $d_p$  jobs in them. We consider algorithm design in the competitive ratio (aka worst-case / adversarial) setting, where the algorithm has no knowledge of any aspect of the arrivals – i.e. it does not know how many packets arrive at each time, and what the deadline and server-subset of each one is. We will allow for both deterministic and randomized algorithms.

We measure the performance of any online allocation algorithm  $\text{alg}$  in terms of its *competitive ratio*  $\rho$ , where

$$\rho(\text{alg}) = \inf_{t > 0} \left( \min_{A \in A_t} \left( \frac{E[S_{\text{alg}}(A)]}{S_{\text{opt}}(A)} \right) \right).$$

where  $A_t$  is the set of all arrival processes such that there are no arrivals

after time-slot  $t$ ,  $E[S_{alg}(A)]$  is the expected number of jobs served within their deadline by the (possibly randomized) algorithm and  $S_{opt}(A)$  is the number of jobs served within their deadline by the offline optimal algorithm, which is the “genie” algorithm that knows the entire sequence a-priori.

## 4.3 Main Results and Discussion

In this section, we state and discuss our main results.

### 4.3.1 Simple Special Case

Before formally describing the algorithm in the next subsection, we provide intuition and illustration by considering a simple arrival process where all jobs arrive at the beginning of time-slot 1 and no further arrivals occur. In addition, we assume that all jobs have a deadline of  $b$  where  $b := d_{max}$  and are revealed to the system sequentially. The scheduling algorithm has to assign a job to a server before the next job arrives, thus making this an online scheduling problem.

Since all jobs have a deadline of  $b$ , to serve all jobs that are queued at a server before their deadlines, we need to ensure that no server is allocated more than  $b$  jobs<sup>2</sup>.

Consider now a convenient representation of the problem, as an *extended* bipartite graph  $(U_b, V, E)$ , defined as follows: for each server  $u \in U$ ,

---

<sup>2</sup>This problem is equivalent to online  $b$ -Matching in bipartite graphs, also studied in [50].

we now make  $b$  copies, to obtain the vertex set  $U_b$  that is one partition of the graph; thus  $|U_b| = b|U|$ . The other partition is just  $V$ , the set of jobs. Recall that each job can only be served by a subset of the servers. Correspondingly, let edge  $(u, v) \in E$  if and only if  $u$  is a copy of a server that can serve  $v$ . An example of this representation, and the algorithm is in Figure 4.2.

The advantage of the extended graph representation is that we have now converted the online allocation problem into one of simple online matching (i.e. one job to one (copy of) a server) on the extended graph. In particular, recall that in our system model, the vertices in  $V$  arrive in arbitrary order, and need to be allocated on arrival. This is exactly the same as finding a matching node  $u \in U_b$  in the extended graph; the fact that each server has at most  $b$  copies ensures the maximum loading is never exceeded.

With this setting, our algorithm first chooses a permutation  $\pi_b$  of  $U_b$  uniformly at random from all possible permutations. This permutation, which we call the *ranking*, is chosen at the beginning and *fixed* thereafter. Note that if  $(u, v) \in E$ ,  $v$  has an edge to all copies of  $u$  in  $\pi_b$ . As each vertex in  $V$  is revealed sequentially, it is matched to the highest ranked unmatched vertex in  $\pi_b$  that it has an edge to.

**Note:** every vertex in  $v$  chooses its vertex according to the same ranking; indeed once this *ranking* is picked, the rest of the algorithm is deterministic. This is what we mean when we say choices are made in a **correlated** random way; the fixing of a single ranking governs the choices of all jobs.

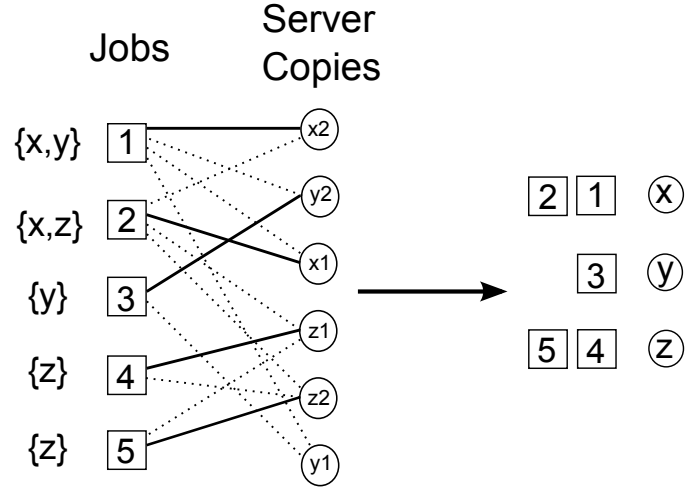


Figure 4.2: An illustration of our algorithm for the simple case of arrivals only in time 1, with all deadlines being  $d_{max} = b = 2$ . Here the server set is  $U = \{x, y, z\}$  and the job set is  $V = \{1, 2, 3, 4, 5\}$ . In the extended graph, there are 2 copies of each server, so  $U_b = \{x_1, y_1, z_1, x_2, y_2, z_2\}$ . Our algorithm picks a random permutation, i.e. ranking,  $\pi_b$  of this set  $U_b$  and fixes it, as shown on the left. Each vertex in set  $V$  is then matched to the highest ranked available vertex in  $U_b$ ; this results in the matching on the extended graph, on the left. The figure on the right shows the collapsing back from extended graph to a server allocation; it shows the resulting allocation with max load of 2 on each server.

The following proposition characterizes the performance of the algorithm in terms of fraction of jobs served, for this simple special case.

**Proposition 1.** *For any set of arrivals in accordance with the special case described above – i.e. all jobs arrive in some order in time slot 1 with deadlines  $b = d_{max}$  – let  $V^*$  be the number of jobs that can be served by the offline optimal “genie” algorithm. Then the number of jobs served by our algorithm is at least*

$$\sum_{i=1}^{V^*} \left(1 + \frac{1}{V^*}\right)^{-i}.$$

*As  $V^* \rightarrow \infty$ , the competitive ratio – i.e. the above quantity divided by  $V^*$  – becomes  $1 - \frac{1}{e}$ .*

Note that, as mentioned, this immediately implies the same competitive ratio for the graph-theoretic problem of online  $b$ -matching, resolving an open issue described in [50].

### 4.3.2 Upper Bound

We now consider again the general case, where arrivals occur over time, and deadlines are completely arbitrary (but bounded by  $d_{max}$ ). In this section we present an upper bound on the competitive ratio of any (possibly randomized) online allocation algorithm; in fact we prove a stronger result, that this worst case can be achieved even by sequences with homogenous deadlines, where every job has a deadline equal to  $d_{max}$ , for any  $d_{max}$ .

The upper bound serves the dual purpose of showing the optimality

of our algorithm, and the gap to optimality of other (more intuitive) online allocation algorithms.

**Theorem 12.** *As the number of servers  $n \rightarrow \infty$ , the competitive ratio  $\rho$ , of any online load balancing algorithm for jobs with strict deadlines is  $\leq 1 - \frac{1}{e}$ .*

Note that the competitive ratio is defined by a minimum over arrival sequences. The above is stronger because it says the minimum even over the restricted set – with homogenous deadlines – is no better, no matter what the value of this homogenous deadline.

### 4.3.3 Our Algorithm and its Performance

The algorithm consists of two parts: a dispatcher and a scheduler. The dispatcher decides which server will serve a job in an online manner and a scheduler decides which job in its queue is served by a server in a given time-slot.

Conventional wisdom dictates that each job should be dispatched to one of the servers which can serve it and has the smallest queue, however, we show that the performance of the naive join the shortest queue algorithm is suboptimal (Theorem 14).

We now describe, first in words and then formally, our online algorithm INSERT RANKING. In each time-slot, the algorithm implements a series of steps as shown in Figure 4.3.

Let  $S$  be the set of servers. Recall that in every time slot, we have a

sequence of jobs arriving; each job  $p$  reveals a subset of servers  $S_p \subseteq S$  that can serve it, a deadline  $d_p$ , and has to be allocated in an online manner.

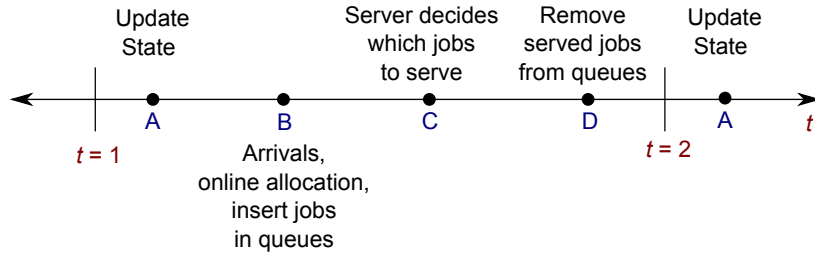


Figure 4.3: INSERT RANKING: Time-line



In the first time-slot,

- A. INITIALIZE: Generate  $d_{max}$  labeled copies of each server with labels 1, 2, ..  $d_{max}$ . Let  $s_i^{(j)}$  denote the copy of  $s_i \in S$  labeled  $j$ . Each labeled server copy  $s_i^{(j)}$  has a value  $V_i^{(j)}$  associated with it, chosen independently according to the uniform distribution on  $[0, 1]$ . Refer to Figure 4.4 for an illustrative example.

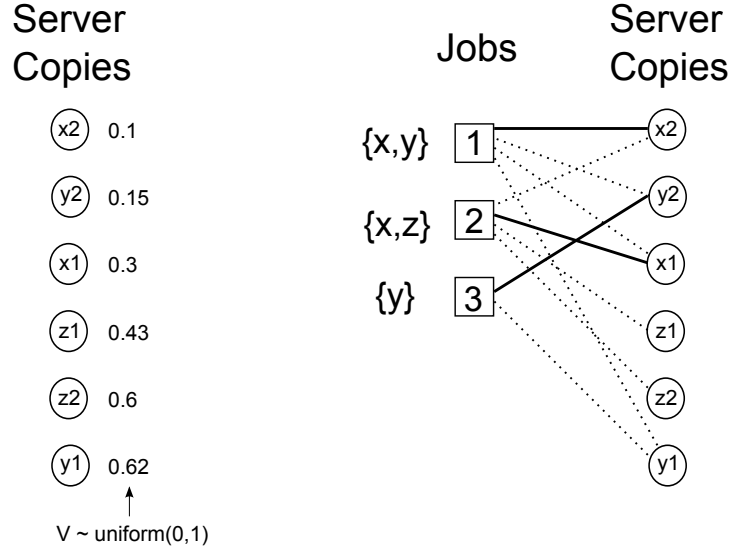


Figure 4.4: An illustration of INSERT RANKING. Here the server set is  $U = \{x, y, z\}$  and  $d_{max} = 2$ . Let all 3 incoming jobs have a deadline of 2 time-slots. The figure on the left is an illustration of INITIALIZE and the figure on the right is an illustration of ONLINE ALLOCATION for time-slot 1.

- B. ONLINE ALLOCATION: All jobs arriving in time-slot 1 are matched to server copies in an online manner as follows: For each incoming job,
1. create the neighborhood of a job  $p$ , defined as all server copies of

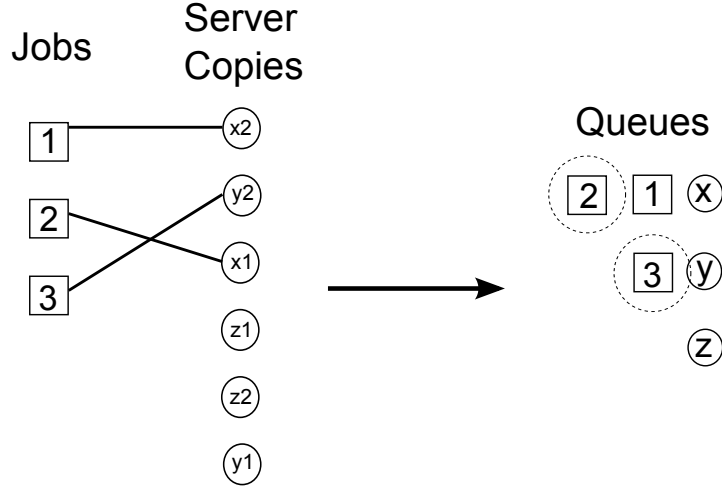


Figure 4.5: An illustration of SCHEDULE for time-slot 1. Server  $x$  serves job 2 since it was matched to  $x1$ , server  $y$  servers job 3, server  $z$  remains idle.

each  $s_i$  in its server subset with label less than  $d_p$  (the label  $j$  of a server copy indicates that a job assigned to that server copy has to be served before the end of time-slot  $j$ ). Therefore, each job  $p$  defined by its server subset and deadline,  $\{S_p, d_p\}$  arriving at time 1 can be served by each  $s \in S_p$  in time-slots  $1, \dots, d_p$ ,

2. match each incoming job to that unmatched server copy in its neighborhood which has the lowest value  $V$ .

Refer to Figure 4.4 for an illustrative example.

- C. SCHEDULE: At the end of job allocation for time-slot 1, all matched jobs are inserted in the queues of the corresponding servers. Each server  $s_i \in S$  serves that job in its queue which was matched to the copy of

$s_i$  with the smallest label. Refer to Figure 4.5 for an illustrative example.

In every *subsequent* time-slot  $t$  ( $> 1$ ),

A. UPDATE STATE:

1. Remove all matched server copies.
2. Remove all server copies labeled  $t - 1$ .
3. Add one new copy of each server with label  $t + d_{max}$  to the set of unmatched servers. Each new server copy  $S_i^{(t+d_{max})}$  has a value  $V_i^{(t+d_{max})}$  associated with it, chosen independently according to the uniform distribution on  $[0, 1]$ .
4. If the label  $j$  of the server copy matched to the job served by a server in the previous slot ( $t - 1$ ) is greater than  $t - 1$ , add  $s_i^{(j)}$  back to the set of unmatched server copies.

Refer to Figure 4.6 for an illustrative example.

- B. ONLINE ALLOCATION: All jobs arriving in time-slot  $t$  are matched to server copies in an online manner as follows: the neighborhood of a job  $p$  is defined as all server copies of each  $s_i$  in its server subset with label less than  $t + d_p - 1$  and each incoming job is matched to that unmatched server copy in its neighborhood which has the lowest value  $V$ .

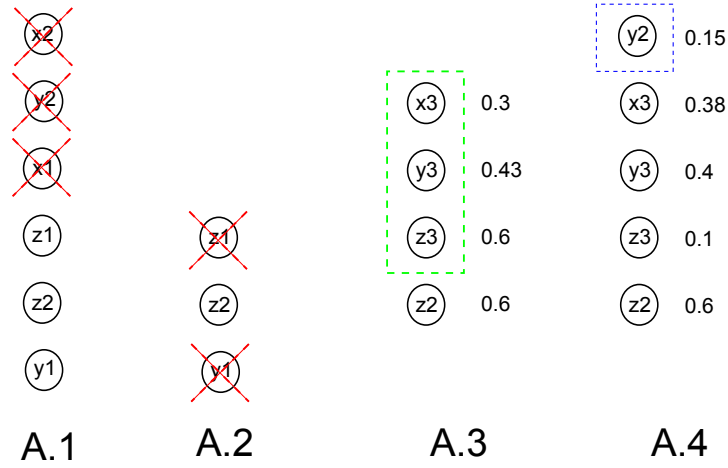


Figure 4.6: An illustration of UPDATE STATE for time-slot 2.

C. SCHEDULE: At the end of job allocation for time-slot  $t$ , all matched jobs are inserted in the queues of the corresponding servers. Each server  $s_i \in S$  serves that job in its queue which was matched to the copy of  $s_i$  with the smallest label.

A more formal definition of the algorithm is presented in Algorithm 1.

---

**Algorithm 1** INSERT RANKING

---

```
1: INITIALIZE:  $t = 1$ ,  $\{s_i^{(j)} : j = t, \dots, t + d_{max} - 1 \text{ and } s_i \in S\}$  and a real
   number  $V_i^{(j)}$  for each  $s_i^{(j)}$  chosen uniformly from  $[0, 1]$ .
   ONLINE ALLOCATION:
2: for arriving job  $p = \{S_p, d_p\}$  in time-slot  $t$ , do
3:   make neighborhood:
      $N(p) = \{s_i^{(j)} : s_i \in S_p \text{ and } t \leq j \leq t + d_p - 1\}$ .
4:   match  $p$  to currently unmatched  $s \in N(p)$  that has the lowest  $V$ .
5: end for
   SCHEDULE:
6: for each server  $s_i$ , do
7:   add all jobs matched to copies of server  $i$  in time-slot  $t$  to the queue of
     server  $s_i$ .
8:   serve currently queued job which was matched to the lowest labeled copy
     of server  $s_i$ .
9: end for
   UPDATE STATE:
10:  $t = t + 1$ .
11: for each server  $s_i$ , do
12:   remove all matched copies of  $s_i$ .
13:   remove  $s_i^{(t-1)}$ .
14:   add  $s_i^{(t+d_{max})}$ , choose a real number  $V_i^{(t+d_{max})}$  uniformly from  $[0, 1]$ .
15:   if  $s_i$  served job  $p$  in time-slot  $t - 1$  and  $p$  was matched to  $s_i^{(j)}$  such that
      $j > t - 1$ , add  $s_i^{(j)}$  back.
16: end for
17: Goto 2.
```

---

**Comments:**

1. INSERT RANKING makes its random choice in a correlated way. The correlated randomness is implemented via the choice of values  $V$  for server copies. Once these values are chosen, the relative priority between two server copies is correlated across jobs in a given slot, and also across

time.

2. INSERT RANKING is NOT a join the shortest queue algorithm. However, it is weighted towards joining shorter queues. We will see in the following subsection that it outperforms natural shortest-queue algorithms.
3. Servers implementing INSERT RANKING are work conserving.
4. Before allocating jobs that arrive in time-slot  $t$ , all server copies with labels  $\leq t - 1$  are removed from the set of unmatched server copies.
5. From the previous comment, it follows that at any given time, there are at most  $nd_{max}$  unmatched server copies, therefore, the storage requirement of INSERT RANKING is  $O(n)$ .

We now prove a lower bound on the competitive ratio of INSERT RANKING.

**Theorem 13.** *Considering all arrivals up to time  $t$ , let  $V^*(t)$  be the number of jobs that can be served by the optimal offline algorithm. As  $V^*(t) \rightarrow \infty$ , the number of jobs served by INSERT RANKING is  $\geq V^*(t)(1 - \frac{1}{e})$ . Therefore, as  $V^*(t) \rightarrow \infty$ , the competitive ratio of INSERT RANKING is  $\geq 1 - \frac{1}{e}$ .*

From Theorem 12 and 13, we conclude that INSERT RANKING is an asymptotically optimal load balancing algorithm.

#### 4.3.4 Performance of other Algorithms

We now analyze the performance of two intuitive randomized algorithms which do not employ correlation in the randomness across jobs. The first obvious candidate algorithm is a join the shortest queue (JSQ) algorithm in which ties are broken in an uniformly random manner. We refer to this algorithm as RANDOMIZED Join the Shortest Queue (RANDOMIZED JSQ). RANDOMIZED JSQ allocates each incoming job to the currently least loaded server (server with the shortest queue) breaking ties uniformly at random independent of past choices. A formal definition of the algorithm is presented in Algorithm 2.

---

**Algorithm 2** RANDOMIZED JSQ

---

```

1: Initialize all QUEUE-LENGTHS  $Q_i$  to 0.
2: for each  $t$  do
3:   for arriving job  $p$  in time-slot  $t$  with server set  $S_p$  and deadline  $d_p$  do
4:      $Q_{min} = \min_{s_i \in S_p} Q_i$ .
5:     if  $Q_{min} < d_p$  then
6:       SHORTEST QUEUES =  $\{s_i : s_i \in S_p, Q_i = Q_{min}\}$ .
7:       Allocate  $p$  to server  $s_j \in$  SHORTEST QUEUES breaking ties uniformly at random independent of all past decisions.
8:       Update QUEUE-LENGTHS.
9:     end if
10:  end for
11:  Each server serves that job in its queue which has the nearest deadline.
    Update  $Q_i$  to  $(Q_i - 1)^+$  for each  $i$ .
12: end for

```

---

**Theorem 14.** *As the number of servers,  $n \rightarrow \infty$ , the competitive ratio of RANDOMIZED JSQ is  $\leq \frac{1}{2}$ .*

We thus conclude that INSERT RANKING strictly outperforms RANDOMIZED JSQ.

INSERT RANKING is not a join the shortest queue algorithm, but, it is biased towards joining shorter queues. To compare it to an algorithm which is similarly biased towards joining shorter queues, but makes its decisions in an uncorrelated manner we propose another algorithm which we refer to as the RANDOMIZED PROBABILISTIC-Join the Shortest Queue (RANDOMIZED P-JSQ) algorithm. RANDOMIZED P-JSQ allocates each incoming job  $p$  with deadline  $d_p$  to server  $s_i$  with probability proportional to  $(d_p - Q_i)^+$  independent of past choices. A formal definition of the algorithm is presented in Algorithm 3.

---

**Algorithm 3** RANDOMIZED P-JSQ

---

```

1: Initialize all QUEUE-LENGTHS  $Q_i$  to 0.
2: for each  $t$  do
3:   for arriving job  $p$  in time-slot  $t$  with server set  $S_p$  and deadline  $d_p$  do
4:     if  $\max_{s_j} (d_p - Q_j) > 0$  then
5:       Allocate  $p$  to server  $s_j \in S_p$  w.p.  $\frac{(d_p - Q_j)^+}{\sum_{s_i \in S_p} (d_p - Q_i)^+}$  independent of
         all past decisions.
6:       Update QUEUE-LENGTHS.
7:     end if
8:   end for
9:   Each server serves that job in its queue which has the nearest deadline.
     Update  $Q_i$  to  $(Q_i - 1)^+$  for each  $i$ .
10: end for

```

---

**Theorem 15.** *As the number of servers,  $n \rightarrow \infty$ , the competitive ratio of RANDOMIZED P-JSQ is  $\leq \frac{1}{2}$ .*



From Theorem 15, we see that INSERT RANKING strictly outperforms RANDOMIZED P-JSQ and we conclude that correlated randomness is key in the good performance of INSERT RANKING.

## 4.4 Simulations

Since all our results are in the limiting case of the number of servers,  $n \rightarrow \infty$ , we present simulation results for finite values of  $n$ . Figure 4.7 shows the performance of INSERT RANKING for the arrival sequence described in Section C.2. Figure 4.8 compares the performance of INSERT RANKING and RANDOMIZED JSQ for the arrival sequence described in Section C.4.1. Figure 4.9 compares the performance of INSERT RANKING and RANDOMIZED P-JSQ for the arrival sequence described in Section C.4.2. In all the plots, we plot the average value of the fraction of jobs served, averaged over 1000 instances of the algorithms.

## 4.5 Summary and Discussion

We study online load balancing under graph constraints in the adversarial setting. First, we design “bad” arrival patterns and use them to upper bound the performance any online load balancing algorithm. Next, we propose an algorithm called INSERT RANKING which uses correlated randomness for load balancing, and prove that it is an optimal online load balancing algorithm. The main message of this chapter is that correlated randomness is important, because we show that INSERT RANKING outperforms algorithms

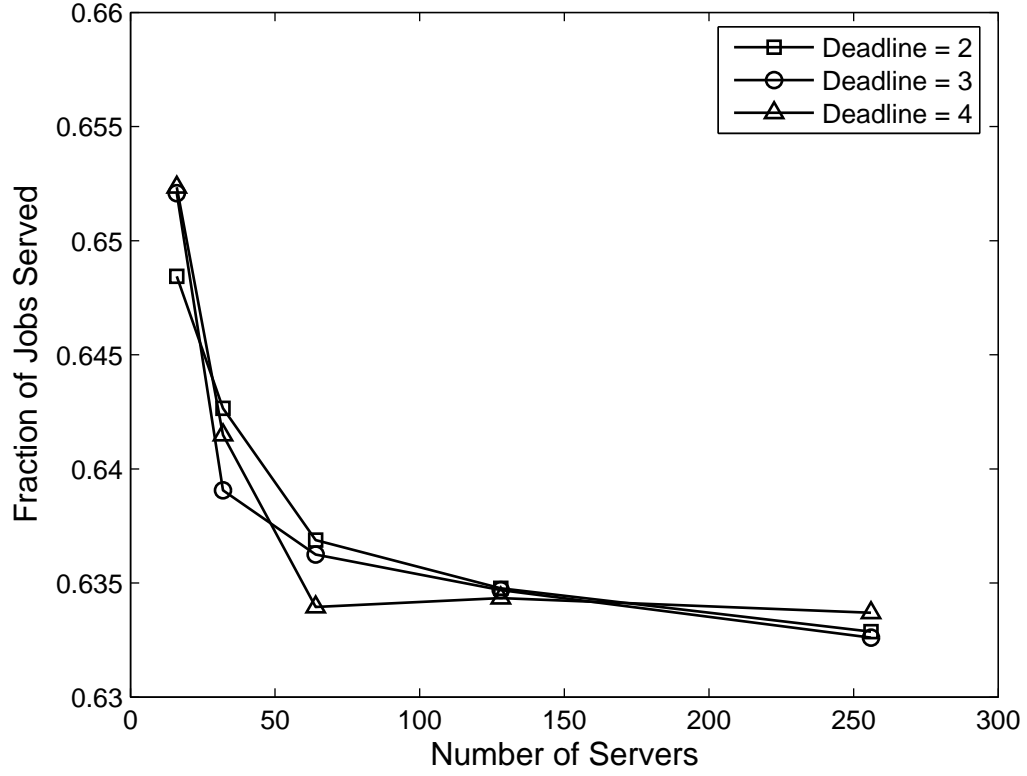


Figure 4.7: INSERT RANKING on the matrix A in Section C.2

which make (the more natural) un-correlated/independent random choices for load balancing.

Graph constrained load balancing problems are of interest for content farms where every content piece cannot be stored on every server. It would also be interesting to look at joint problem of predicting/infering popularity, placing popular items on servers in content farms, and serving the resulting demand.

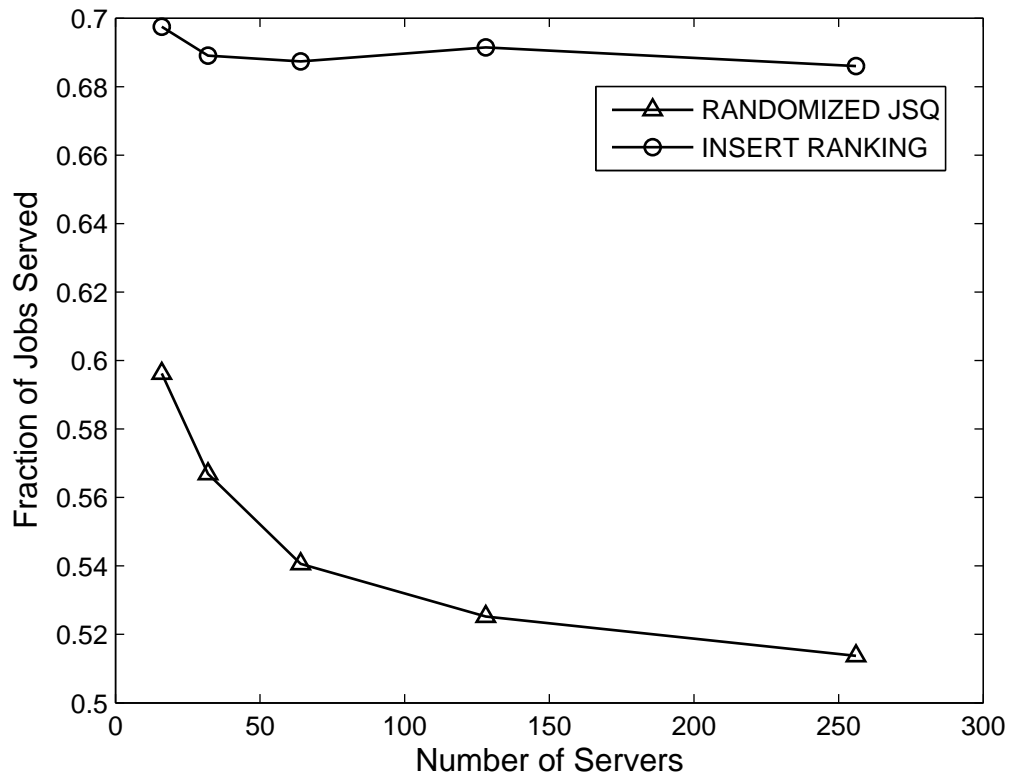


Figure 4.8: Comparison of INSERT RANKING and RANDOMIZED JSQ

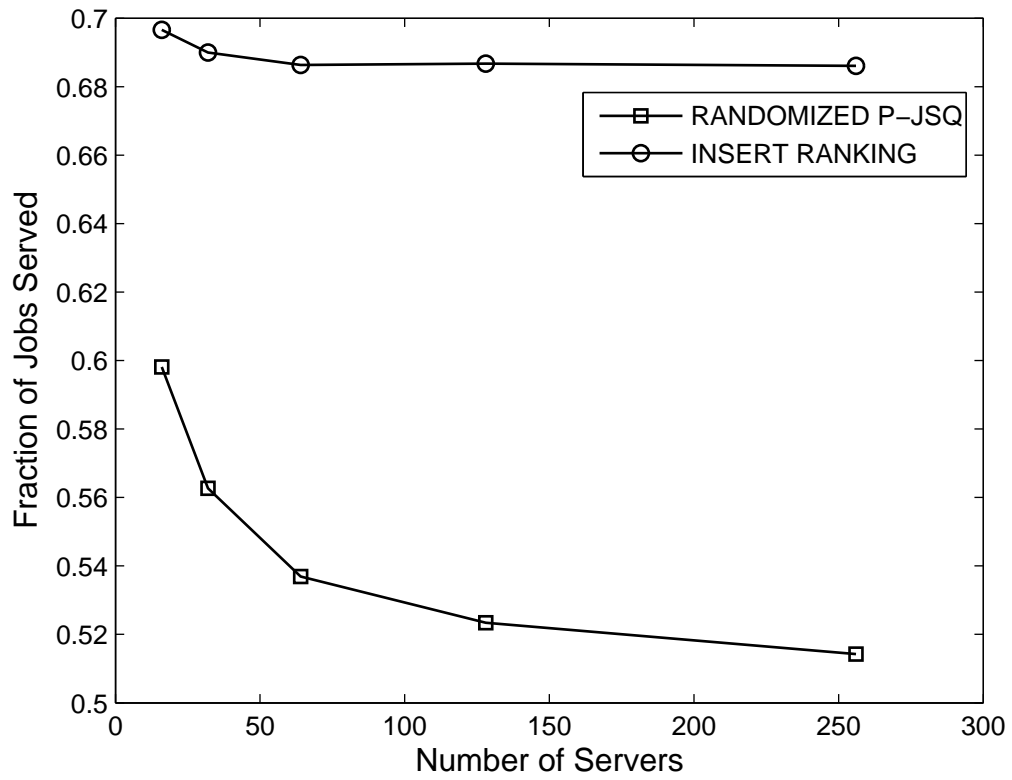


Figure 4.9: Comparison of INSERT RANKING and RANDOMIZED P-JSQ

## Chapter 5

# Serving Content with Unknown Demand: the High-Dimensional Regime

### 5.1 Introduction

Ever increasing volumes of multimedia content is now requested and delivered over the Internet.<sup>1</sup> Content delivery systems (e.g., YouTube [109]), consisting of a large collection of servers (each with limited storage/service capability), process and service these requests. Naturally, the storage and content replication strategy (i.e., what content should be stored on each of these servers) forms an important part of the service and storage architecture.

Two trends have emerged in such settings of large-scale distributed content delivery systems. First, there has been a sharp rise in not just the volume of data, but indeed in *the number of content-types* (e.g., number of distinct YouTube videos) that are delivered to users [109]. Second, the popularity and demand for most of this content is *uneven and ephemeral*; in many cases, a particular content-type (e.g., a specific video clip) becomes popular for a small

---

<sup>1</sup>S. Moharir, J. Ghaderi, S. Sanghavi, and S. Shakkottai. “Serving content with unknown demand: the high-dimensional regime.” In proceedings of the ACM international conference on Measurement and modeling of computer systems (SIGMETRICS 2014). The coauthors on the paper had equal contributions in obtaining these results.

interval of time after which the demand disappears; further a large fraction of the content-types languish in the shadows with almost no demand [34, 5].

To understand the effect of these trends, we study a stylized model for the content placement and delivery in large-scale distributed content delivery systems. The system consists of  $n$  servers, each with constant storage and service capacities, and  $\alpha n$  content-types ( $\alpha$  is some constant number). We consider the scaling where the system size  $n$  tends to infinity. The requests for the content-types arrive dynamically over time and need to be served in an online manner by the free servers storing the corresponding contents. The requests that are “deferred” (i.e., cannot be immediately served by a free server with requested content-type) incur a high cost. To ensure reliability, we assume that there are alternate server resources (e.g., a central server with large enough backup storage and service capacity, or additional servers that can be freed up on-demand) that can serve such deferred requests.

The performance of any content placement strategy crucially depends on the popularity distribution of the content. Empirical studies in many services such as YouTube, Peer-to-Peer (P2P) VoD systems, various large video streaming systems, and web caching, [34, 17, 110, 43, 95] have shown that access for different content-types is very inhomogeneous and typically matches well with power-law (Zipf-like) distributions, i.e., the request rate for the  $i$ -th most popular content-type is proportional to  $i^{-\beta}$ , for some parameter  $\beta > 0$ . For the performance analysis, we assume that the content-types have a popularity that is governed by some power-law distribution with unknown  $\beta$  and

further this distribution changes over time.

Our objective is to provide efficient content placement strategies that minimize the number of requests deferred. It is natural to expect that content placement strategies in which more popular content-types are replicated more will have a good performance. However, there is still a lot of flexibility in designing such strategies and the extent of replication of each content-type has to be determined. Moreover, the requests arrive dynamically over time and popularities of different content-types might vary significantly over time; thus the content placement strategy needs to be online and robust.

The fact that the number of contents is very large and their popularities are time-varying creates two new challenges that are not present in traditional queueing systems. First, it is imperative to *measure the performance of content replication strategies over the time scale in which changes in popularities occur*. In particular, the steady-state metrics typically used in queueing systems are not a right measure of performance in this context. Second, the number of content-types is enormous and *learning the popularities of all content-types over the time scale of interest is infeasible*. This is in contrast with traditional multi-class multi-server systems where the number of demand classes does not scale with the number of servers (low-dimensional setting) and thus learning the demand rates can be done in a time duration that does not scale with the system size.

### 5.1.1 Contributions

The main contributions of our work can be summarized as follows.

**Modeling Contribution:** We recognize that we are in the high-dimensional regime with unknown demand, that it is fundamentally different from the low-dimensional setting (finite number of content-types) and propose a model that captures this difference.

**Analytical Contributions:** In Section 5.3.1, we show that in this high-dimensional setting where the demand statistics are not known a-priori, the “*learn-and-optimize*” approach, i.e., learning the demand statistics from requests and then locally caching content on servers using the estimated statistics, is strictly sub-optimal, even when using high-dimensional estimators such as the Good-Turing estimator [66] (Theorem 16). This is in contrast to the conventional low-dimensional setting where the “learn-and-optimize” approach is asymptotically optimal.

In addition, in Section 5.3.2, we study an adaptive content replication strategy which myopically attempts to cache the most recently requested content-types on idle servers. Our key result is that even this simple adaptive strategy strictly outperforms *any* content placement strategy based on the “learn-and-optimize” approach (Theorem 18). Our results also generalize to the setting where the demand statistics change with time (Theorems 17 and 19).

Overall, our results demonstrate that separating the estimation of de-



mands and the subsequent use of the estimations to design optimal content placement policies is deprecated in the high-dimensional setting.

### 5.1.2 Organization and Basic Notations

The rest of this chapter is organized as follows. We describe our system model and setting in Section 5.2. The main results are presented in Section 5.3. Our simulation results are discussed in Section 6.4. The proofs of our results are discussed in the Appendix. Section 5.5 gives an overview of related works. We finally end the chapter with conclusions.

Some of the basic notations are as follows. Given two functions  $f$  and  $g$ , we write  $f = O(g)$  if  $\limsup_{n \rightarrow \infty} |f(n)/g(n)| < \infty$ .  $f = \Omega(g)$  if  $g = O(f)$ . If both  $f = O(g)$  and  $f = \Omega(g)$ , then  $f = \Theta(g)$ . Similarly,  $f = o(g)$  if  $\limsup_{n \rightarrow \infty} |f(n)/g(n)| = 0$ , and  $f = \omega(g)$  if  $g = o(f)$ . The term *w.h.p.* means with high probability as  $n \rightarrow \infty$ .

## 5.2 Setting and Model

In this section, we consider a stylized model for large scale distributed content systems that captures two emerging trends, namely, a large number of content types, and uneven and time-varying demands.

### 5.2.1 Server and Storage Model

The system consists of  $n$  front-end servers, each of which can hold one content piece, and serve one user, at any time. In addition, there is a back-

end server that stores the entire catalog of  $m$  content-types (one copy of each content-type, e.g., a copy of each YouTube video). The contents can be copied from the back-end server and placed on the front-end servers.

Since we are interested in the scaling performance, as  $n, m \rightarrow \infty$ , for clarity we assume that there are  $n$  servers and each server can store 1 content and can serve 1 request at any time. If instead of one content, each front-end server can store at most  $d > 1$  content pieces ( $d$  is a constant) and serve at most  $d$  requests at each time, the performance can be bounded from above by the performance of a system with  $dn$  servers with a storage of 1 each, and from below by that of another system with  $n$  servers with a storage of 1 each. Thus asymptotically in a scaling-sense, the system is still equivalent to a system of  $n$  servers where each server can store 1 content and can serve 1 content request at any time.

### 5.2.2 Service Model

When a request for a content arrives, it is routed to an idle (front-end) server which has the corresponding content-type stored on it, if possible. We assume that the service time of each request is exponentially distributed with mean 1. The requests have to be served in an online manner; further service is non-preemptive, i.e., once a request is assigned to a server, its service cannot be interrupted and also cannot be re-routed to another server. Requests that cannot be served (no free server with requested content-type) incur a high cost (e.g., need to be served by the back-end server, or content needs to be fetched

from the back-end server and loaded on to a new server). As discussed before, we refer to such requests as deferred requests. The goal is to design content placement policies such that the number of requests deferred is minimized.

### 5.2.3 Content Request Model

There are  $m$  content-types (e.g.,  $m$  distinct YouTube videos). We consider the setting where the number of content-types  $m$  is very large and scales linearly with the system size  $n$ , i.e.,  $m = \alpha n$  for some constant  $\alpha > 1$ . We assume that requests for each content arrive according to a Poisson process and request rates (popularities) follow a Zipf distribution. Formally, we make the following assumptions on the arrival process.

*Assumption (i).* (Arrival and Content Request Process)

- The arrival process for each content-type  $i$  is a Poisson process with rate  $\lambda_i$ .
- The load on the system at any time is  $\bar{\lambda} < 1$ , where  $\bar{\lambda} = \frac{\sum_{i=1}^m \lambda_i}{n}$ .
- Without loss of generality, content-types are indexed in the order of popularity. The request rate for content-type  $i$  is  $\lambda_i = n\bar{\lambda}p_i$  where  $p_i \propto i^{-\beta}$  for some  $\beta > 0$ . This is the Zipf distribution with parameter  $\beta$ .

We have used the Zipf distribution to model the popularity distribution of various contents because empirical studies in many content delivery systems have shown that the distribution of popularities matches well with such distributions, see e.g., [34], [17], [110], [43], [95].

### 5.2.4 Time Scales of Change in Arrival Process

A key trend discussed earlier is the time-varying nature of popularities in content delivery systems [34, 5]. For example, the empirical study in [34] (based on 25 millions transactions on YouTube) shows that daily top 100 list of videos frequently changes. To understand the effect of this trend on the performance of content placement strategies, we consider the following two change models.

**Block Change Model:** In this model, we assume that the popularity of various content-types remains constant for some duration of time  $T(n)$ , and then changes to some other arbitrarily chosen distribution that satisfies Assumption (i). Thus  $T(n)$  reflects the time-scale over which changes in popularities occur. Under this model, we characterize the performance of content placement strategies over such a time-scale  $T(n)$ .

**Continuous Change Model:** Under this model, we assume that each content-type has a Poisson clock at some constant rate  $\nu > 0$ . Whenever the clock of content-type  $i$  ticks, content-type  $i$  exchanges its popularity with some other content-type  $j$ , chosen uniformly at random. Note that the average time over which the popularity distribution “completely” changes is comparable to that of the Block Change Model; however, here the change occurs incrementally and continuously. Note that this model ensures that the content-type popularity always has the Zipf distribution. Under this model, we characterize the

performance of content placement strategies over constant intervals of time.

### 5.3 Main Results and Discussion

In this section, we state and discuss our main results. The proofs are provided in the Appendix.

#### 5.3.1 Separating Learning from Content Placement

In this section, we analyze the performance of storage policies which separate the task of learning and that of content placement as follows. Consider time intervals of length  $T(n)$ . The operation of the policy in each time interval is divided into two phases:

**Phase 1. Learning:** Over this interval of time, use the demands from the arrivals (see Figure 5.1) to estimate the content-type popularity statistics.

**Phase 2. Storage:** Using the estimated popularity of various content-types, determine which content-types are to be replicated and stored on each server. The storage is fixed for the remaining time interval. The content-types not requested even once in the learning phase are treated equally in the storage phase. In other words, the popularity of all *unseen* content-types in the learning phase is assumed to be the same.

Further, we allow the interval of time for the Learning phase poten-

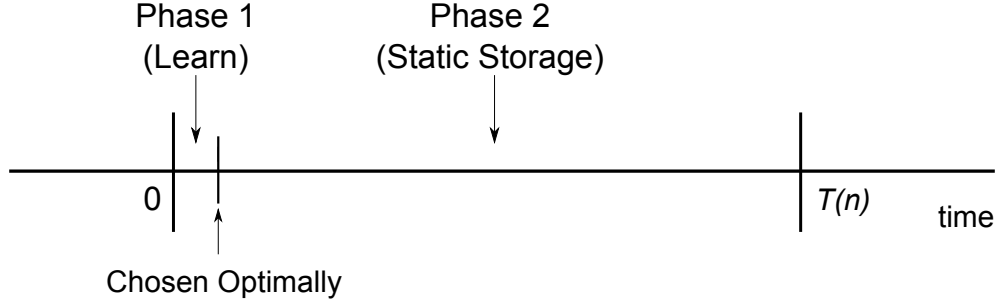


Figure 5.1: Learning-Based Static Storage Policies – *The interval  $T(n)$  is split into the Learning and Storage phases. The length of time spent in the Learning phase can be chosen optimally using the knowledge of the value of  $T(n)$  and the Zipf parameter  $\beta$ .*

tially to be chosen optimally using knowledge of  $T(n)$  (the interval over which statistics remain stationary) and  $\beta$  (the Zipf parameter for content-types popularity).

This is a natural class of policies to consider because it is obvious that popular content-types should be stored on more servers than the less popular content-types. Therefore, knowing the arrival rates can help in the design of better storage policies. Moreover, for the content-types which are not seen in the learning phase, the storage policy has no information about their relative popularity. It is therefore natural to treat them as if they are equally popular.

The replication and storage in Phase 2 (Storage) can be performed by *any* static policy that relies on the knowledge (estimate) of arrival rates, e.g., the proportional placement policy [58] where the number of copies of each content-type is proportional to its arrival rate, or the storage policy of [59] which was shown to be approximately optimal in the steady state.

We now analyze the performance of learning-based static storage policies under the Block Change Model defined in Section 5.2.4 where the statistics remain invariant over the time intervals of length  $T(n)$ . The performance metric of interest is the number of requests deferred by any policy belonging to class of learning-based static storage policies in the interval of interest. We assume that at the beginning of this interval, the storage policy has no information about the relative popularity of various content-types. Therefore, we start with an initial loading where each content-type is placed on exactly one server. This loading is not changed during Phase 1 (the learning phase) at the end of which, the content-type on idle servers is changed as per the new storage policy. As mentioned before, this storage is not changed for the remaining duration in the interval of interest.

The following theorem provides a lower bound on the number of requests deferred by any learning-based static storage policy for the Block Change Model.

**Theorem 16.** *Under Assumption (i) and the Block Change Model defined in Section 5.2.4, for  $\beta > 1$ , if  $T(n) = \Omega(1)$ , the expected number of requests deferred by any learning-based static storage policy is  $\Omega(\min\{(nT(n))^{\frac{1}{2-1/\beta}}, n\})$ .*

We therefore conclude that even if the division of the interval of interest into Phase 1 (Learning) and Phase 2 (Storage) is done in the optimal manner, no learning-based static storage policy can defer fewer than  $\Omega((nT(n))^{\frac{1}{2-1/\beta}})$  jobs in the interval of interest. Therefore, Theorem 16 provides a fundamental

lower bound on the number of jobs deferred by any policy which separates learning and storage. It is worth pointing out that this result holds even when the time-scale of change in statistics is quite slow. Thus, even when  $T(n)$ , the time-scale over which statistics remains invariant, goes to infinity and the time duration of the two phases (Learning, Storage) is chosen optimally based on  $\beta$ ,  $T(n)$ ,  $\Omega(\min\{(nT(n))^{\frac{1}{2-1/\beta}}, n\})$  requests are still deferred.

The next theorem provides a lower bound on the number of requests deferred by any learning-based static storage policy for the Continuous Change Model. As before, we assume that at the beginning of this interval, the storage policy has no information about content popularity and therefore, we start with an initial loading where each content-type is placed on exactly one server.

**Theorem 17.** *Under Assumption (i) and the Continuous Change Model defined in Section 5.2.4, for  $\beta > 1$ , if  $T(n) = \Omega(1)$ , the expected number of requests deferred by any learning-based static storage policy is*

$$\Omega(\min\{(nT(n))^{\frac{1}{2-1/\beta}}, n\}).$$

Next, we explore *adaptive storage policies* which perform the task of learning and storage simultaneously.

### 5.3.2 Myopic Joint Learning and Placement

We next study a natural adaptive storage policy called MYOPIC. In an adaptive storage policy, depending on the requests that arrive and depart, the content-type stored on a server can be changed when the server is idle



while other servers of the system might be busy serving requests. Therefore, adaptive policies perform the tasks of learning and placement jointly. Many variants of such adaptive policies have been studied for decades in the context of cache management (e.g. LRU, LRU-MIN [102]).

Let  $C_i$  refer to the  $i^{th}$  content-type,  $1 \leq i \leq m$ . The MYOPIC policy works as follows: When a request for content-type  $C_i$  arrives, it is assigned to a server if possible, or deferred otherwise. Recall that a deferred request is a request for which on arrival, no currently idle server can serve it and thus its service invokes a backup mechanism such as a back-end server which can serve it at a high cost. After the assignment/defer decision is made, if there are no currently idle servers with content-type  $C_i$ , MYOPIC replaces the content-type of one of the idle servers with  $C_i$ . This idle server is chosen as follows:

- If there is a content-type  $C_j$  stored on more than one currently idle server, the content-type of one of those servers is replaced with  $C_i$ ,
- Else, place  $C_i$  on that currently idle server whose content-type has been requested least recently among the content-types on the currently idle servers.

For a formal definition of MYOPIC, refer to Figure 5.2.

*Remark 1.* Some key properties of MYOPIC are:

1. The content-types on servers can be potentially changed only when there is an arrival.

---

```

1: On arrival (request for  $C_i$ ) do,
2:   Allocate request to an idle server if possible.
3:   if no other idle server has a copy of  $C_i$ , then
4:     if  $\exists j$ :  $C_j$  stored on  $> 1$  idle servers, then
5:       replace  $C_j$  with  $C_i$  on any one of them.
6:     else
7:       find  $C_j$ : least recently requested on idle servers,
       replace  $C_j$  with  $C_i$ .
8:     end if
9:   end if

```

---

Figure 5.2: MYOPIC – An adaptive storage policy which changes the content stored on idle servers in a greedy manner to ensure that recently requested content pieces are available on idle servers.

2. The content-type of at most one idle server is changed after each arrival.

However, for many popular content-types, it is likely that there is already an idle server with the content-type, in which case there is no content-type change.

3. To implement MYOPIC, the system needs to keep track of the time at which the recent most request of each content-type was made.

The following theorem provides an upper bound on the number of requests deferred by MYOPIC for the Block Change Model defined in Section 5.2.4.

**Theorem 18.** *Under Assumption (i) and the Block Change Model defined in Section 5.2.4, over any time interval  $T(n)$  such that  $T(n) = o(n^{\beta-1})$ , the number of requests deferred by MYOPIC is  $O((nT(n))^{1/\beta})$  w.h.p.*

We now compare this upper bound with the lower bound on the number of requests deferred by any learning-based static storage policy obtained in Theorem 16.

**Corollary 1.** *Under Assumption (i), the Block Change Model defined in Section 5.2.4, and for  $\beta > 1$ , over any time interval  $T(n)$  such that  $T(n) = \Omega(1)$  and  $T(n) = o(n^{\beta-1})$ , the expected number of requests deferred by any learning-based static storage policy is  $\Omega(\min\{(nT(n))^{\frac{1}{2-1/\beta}}, n\})$  and the number of requests deferred by the MYOPIC policy is  $O((nT(n))^{\frac{1}{\beta}})$  w.h.p.*

For  $\beta > 1$ ,  $\frac{1}{2-1/\beta} > \frac{1}{\beta}$  and for  $T(n) = o(n^{\beta-1})$ ,  $(nT(n))^{\frac{1}{\beta}} = o(n)$ . Therefore, from Corollary 1, we conclude that MYOPIC outperforms all learning-based static storage policies. Note that:

- i. Corollary 1 holds even when the interval of interest  $T(n)$  grows to infinity (scaling polynomially in  $n$ ), or correspondingly, even when the content-type popularity changes very slowly with time.
- ii. Even if the partitioning of the  $(T(n))$  into a Learning phase and a Static Storage phase is done in an optimal manner with the help of some side information  $(\beta, T(n))$ , the MYOPIC algorithm outperforms any learning-based static storage policy.
- iii. Since we consider the high-dimensional setting, the learning problem at hand is a large-alphabet learning problem. It is well known that standard estimation techniques like using the empirical values as estimates of the

true statistics is suboptimal in this setting. Many learning algorithm like the classical Good-Turing estimator [66] and other linear estimators [94] have been proposed, and shown to have good performance for the problem of large-alphabet learning. From Corollary 1, we conclude that, even if the learning-based storage policy uses the best possible large-alphabet estimator, it cannot match the performance of the MYOPIC policy.

Therefore, in the high-dimensional setting we consider, separating the task of estimation of the demand statistics, and the subsequent use of the same to design a static storage policy, is strictly suboptimal. This is the key message of this chapter.

Theorem 18 characterizes the performance of MYOPIC under the Block Change Model, where the statistics of the arrival process do not change in interval of interest. To gain further insight into robustness of MYOPIC against changes in the arrival process, we now analyze the performance of MYOPIC when the arrival process can change in the interval of interest according to the Continuous Change Model defined in Section 5.2.4.

Recall that under the Continuous Change Model, on average, we expect  $\Theta(n)$  shuffles in the popularity of various content-types in an interval of constant duration. For the Block Change Model, if  $T(n) = \Theta(1)$ , the entire popularity distribution can change at the end of the block, which is equivalent

to  $n$  shuffles. Therefore, for both the change models, the expected number of changes to the popularity distribution in an interval of constant duration is of the same order. However, these changes occur constantly but slowly in the Continuous Change Model as opposed to a one-shot change in the Block Change Model.

**Theorem 19.** *Under Assumption (i), and the Continuous Change Model defined in Section 5.2.4, the number of requests deferred by the MYOPIC storage policy in any interval of constant duration is  $O(n^{1/\beta})$  w.h.p.*

In view of Theorem 18, if the arrival rates do not vary in an interval of constant duration, under the MYOPIC storage policy, the number of requests deferred in that interval is  $O(n^{1/\beta})$  w.h.p. Theorem 19 implies that the number of requests deferred in a constant duration interval is of the same order even if the arrival rates change according to the Continuous Change Model. This shows that the performance of the MYOPIC policy is robust to changes in the popularity statistics.

We now compare the upper bound obtained in Theorem 19 for the Continuous Change Model with the lower bound on the performance of any learning-based static storage policy obtained in Theorem 17.

**Corollary 2.** *Under Assumption (i), the Continuous Change Model defined in Section 5.2.4, and for  $\beta > 1$ , over any time interval of constant duration, the expected number of requests deferred by any learning-based static storage*

*policy is  $\Omega(n^{\frac{1}{2-1/\beta}})$  and the number of requests deferred by the MYOPIC policy is  $O(n^{\frac{1}{\beta}})$  w.h.p.*

Thus, even for the Continuous Change Model, MYOPIC outperforms all Learning-based static policies. Compared to the Block Change Model, Learning-based static policies are “unsuitable” for the Continuous Change Model due to the following reasons:

- Content popularity can change while the system is in the learning phase. This makes the task of estimating content popularity more difficult.
- Once storage is optimized for the estimated content popularity (at the end of Phase 1), it is not changed in Phase 2. However, content popularities will change (by a small amount) almost instantaneously after the learning period, thus making the storage suboptimal even if content popularity was estimated accurately in Phase 1.

### **5.3.3 Genie-Aided Optimal Storage Policy**

In this section, our objective is to study the setting where the demand statistics are available “for free”. For the Block Change Model with known popularity statistics, we show that a simple adaptive policy is optimal in the class of all policies which know popularity statistics of various content-types. We denote the class of such policies as  $\mathbb{A}$  and refer to the optimal policy as the GENIE policy.

Let the content-types be indexed from  $i = 1$  to  $m$  and let  $C_i$  be the  $i^{th}$  content-type. Without loss of generality, we assume that the content-types are indexed in the order of popularity, i.e,  $\lambda_i \geq \lambda_{i+1}$  for all  $i \geq 1$ . Let  $k(t)$  denote the number of idle servers at time  $t$ .

The key idea of the GENIE storage policy is to ensure that at any time  $t$ , if the number of idle servers is  $k(t)$ , the  $k(t)$  most popular content-types are stored on exactly one idle server each. The GENIE storage policy can be implemented as follows. Recall  $C_i$  is the  $i^{th}$  most popular content-type. At time  $t$ ,

- If there is a request for content-type  $C_i$  with  $i < k(t^-)$ , then allocate the request to the corresponding idle server. Further, replace the content-type on server storing  $C_{k(t^-)}$  with content-type  $C_i$ .
- If there is a request for content-type  $C_i$  with  $i > k(t^-)$ , defer this request. There is no storage update.
- If there is a request for content-type  $C_i$  with  $i = k(t^-)$ , then allocate the request to the corresponding idle server. There is no storage update.
- If a server becomes idle (due to a departure), replace its content-type with  $C_{k(t^-)+1}$ .

For a formal definition, please refer to Figure 5.3.

---

```

1: Initialize: Number of idle-servers :=  $k = n$ .
2: while true do
3:   if new request (for  $C_i$ ) routed to a server, then
4:     if  $i \neq k$ , then
5:       replace content-type of idle server storing  $C_k$  with  $C_i$ 
6:     end if
7:      $k \leftarrow k - 1$ 
8:   end if
9:   if departure, then
10:    replace content-type of new idle server with  $C_{k+1}$ 
11:     $k \leftarrow k + 1$ 
12:   end if
13: end while

```

---

Figure 5.3: GENIE – An adaptive storage policy which has content popularity statistics available for “free”. At time  $t$ , if the number of idle servers is  $k(t)$ , the  $k(t)$  most popular content-types are stored on exactly one idle server each.

*Remark 2.* The implementation of GENIE requires replacing the content-type of at most one server on each arrival and departure.

To characterize the performance of GENIE, we assume that the system starts from the empty state (all servers are idle) at time  $t = 0$ . The performance metric for any policy  $\mathcal{A}$  is  $D^{(\mathcal{A})}(t)$ , defined as the number of requests deferred by time  $t$  under the adaptive storage policy  $\mathcal{A}$ . We say that an adaptive storage policy  $\mathcal{O}$  is optimal if

$$D^{(\mathcal{O})}(t) \leq_{st} D^{(\mathcal{A})}(t), \quad (5.1)$$

for any storage policy  $\mathcal{A} \in \mathbb{A}$  and any time  $t \geq 0$ . Where Equation 5.1 implies that,

$$\mathbb{P}(D^{(\mathcal{O})}(t) > x) \leq \mathbb{P}(D^{(\mathcal{A})}(t) > x),$$



for all  $x \geq 0$  and  $t \geq 0$ .

**Theorem 20.** *If the arrival process to the content-type delivery system is Poisson and the service times are exponential random variables with mean 1, for the Block Change Model defined in Section 5.2.4, let  $D^{(\mathcal{A})}(t)$  be the number of requests deferred by time  $t$  under the adaptive storage policy  $\mathcal{A} \in \mathbb{A}$ . Then, we have that,*

$$D^{(\text{GENIE})}(t) \leq_{st} D^{(\mathcal{A})}(t),$$

for any storage policy  $\mathcal{A} \in \mathbb{A}$  and any time  $t \geq 0$ .

Note that this theorem holds even if the  $\lambda_i$ s are not distributed according to the Zipf distribution. We thus conclude that GENIE is the optimal storage policy in the class of all storage policies which at time  $t$ , have no additional knowledge of the future arrivals except the values of  $\lambda_i$  for all content-types and the arrivals and departures in  $[0, t)$ . Next, we compute a lower bound on the performance of GENIE.

**Theorem 21.** *Under Assumption (i), for  $\beta > 1$ , the Block Change Model defined in Section 5.2.4 and if the interval of interest is of constant length, the expected number of requests deferred by GENIE is  $\Omega(n^{2-\beta})$ .*

From Theorems 18 and 21 we see that there is a gap in the performance of the MYOPIC policy and the GENIE policy (which has additional knowledge of the content-type popularity statistics). Since for the GENIE policy, learning

the statistics of the arrival process comes for “free”, this gap provides an upper bound on the cost of serving content-type with *unknown* demands. We compare the performance of the all the policies considered so far in the next section via simulations.

As discussed before, the key property of the GENIE storage policy is that at time  $t$ , if there are  $k(t)$  idle servers, the policy ensures that exactly one copy of the  $k(t)$  most popular contents is stored on the idle servers. In Figure 5.3, we describe how to preserve this property at all times, in the setting where content popularity remains constant in the interval of interest. If content popularity is time-varying, as in the case of the Continuous Change Model, to maintain this property, the policy needs to have instantaneous knowledge of any change in content popularity. Moreover, contents stored on idle servers might need to be changed at the instant of change in content popularity to ensure that the idle servers store the currently most popular contents at all times.

Since the MYOPIC and GENIE policies are adaptive policies, contents stored on the front-end servers are changed dynamically. Such content changes can be classified into two types: internal fetches and external fetches. An internal fetch occurs when a content is available on at least one front-end server and the storage policy needs to place a copy of this content on an idle front-end server. In such cases, we assume that the new copy is fetched internally from one of the local (front-end) servers storing this content. An external fetch occurs when the content is currently not stored on any of the front-end

servers (busy/idle) and hence the copy needs to be fetched externally from the back-end server. The external fetches incur a much higher cost compared to the internal fetches as data transfer from outside is subject to high delay and/or bandwidth consumption. The next theorem provides bounds on the number of external fetches performed to implement the MYOPIC and GENIE policies under the Block Change Model. Since the comparison depends on the initial storage of servers at the beginning of the block, we consider the worst initial case for the MYOPIC policy which is an empty system.

**Theorem 22.** *Let  $V^{P^*}(T)$  be the number of external fetches made while implementing the storage policy  $P^*$  in the time-interval  $(0, T)$ . Under Assumption 1, for  $\beta > 1$ , the Block Change Model and assuming we start from an empty system, for  $T = O(1)$ ,*

$$(i) \quad V^{(MYOPIC)}(T) = O(nT)^{1/\beta} \text{ w.h.p.}$$

$$(ii) \quad V^{(GENIE)}(T) = \Omega\{\min\{n, nT\}\} \text{ w.h.p.}$$

Thus the MYOPIC policy incurs fewer external fetches compared to the GENIE policy. This is not surprising as the GENIE storage policy is designed with the objective of minimizing the number of deferred requests, and hence it is more aggressive in changing the contents stored on servers in order to minimize the probability that the next request is deferred.

## 5.4 Simulation Results

We compare the performance of the MYOPIC policy with the performance of the GENIE policy and the following two learning-based static storage policies:

- The “*Empirical + Static Storage*” policy uses the empirical popularity statistics of content types in the learning phase as estimates of the true popularity statistics. At the end of the learning phase, the number of servers on which a content is stored is proportional to its estimated popularity.
- The “*Good Turing + Static Storage*” policy uses the Good-Turing estimator [66] to compute an estimate of the missing mass at the end of the learning phase. The missing mass is defined as total probability mass of the content types that were not requested in the learning phase. Recall that we assume that learning-based static storage policies treat all the missing content-types equally, i.e., all missing content-types are estimated to be equally popular.

Let  $M_0$  be the total probability mass of the content types that were not requested in the learning phase and  $S_1$  be the set of content types which were requested exactly once in the learning phase. The Good-Turing estimator of the missing mass ( $\widehat{M}_0$ ) is given by

$$\widehat{M}_0 = \frac{|S_1|}{\text{number of samples}}.$$

See [66] for details.

Let  $N_i$  be the number of times content  $i$  was requested in the learning phase and  $\mathcal{C}_{\text{missing}}$  be the set of content-types not requested in the learning phase. The “Good Turing + Static Storage” policy computes an estimate of the content-popularity as follows:

- i: If  $N_i = 0$ ,  $p_i = \frac{\widehat{M}_0}{|\mathcal{C}_{\text{missing}}|}$ .
- ii: If  $N_i > 0$ ,  $p_i = (1 - \widehat{M}_0) \frac{N_i}{\text{number of samples}}$ .

At the end of the learning phase, the number of servers on which a content is stored is proportional to its estimated popularity.

We simulate the content distribution system for arrival and service process which satisfy Assumption (i) to compare the performance of the four policies mentioned above and also understand how their performance depends on various parameters like system size ( $n$ ), load ( $\bar{\lambda}$ ) and Zipf parameter ( $\beta$ ). In Tables 5.1, 5.2 and 5.3, we report the mean and variance of the fraction of jobs served by the policies over a duration of 5 s ( $T(n) = 5$ ).

For each set of system parameters, we repeat the simulations between 1000 to 10000 times for each policy in order to ensure that the standard deviation of the quantity of interest (fraction of jobs served) is small and comparable. For the two adaptive policies (GENIE and MYOPIC), the results are averaged over 1000 iterations and for the learning-based policies (“Empirical + Static Storage” and “Good-Turing + Static Storage”), the results are averaged over

10000 iterations. In addition, the results for the learning-based policies are reported for empirically optimized values for the fraction of time spent by the policy in learning the distribution.

In Table 5.1, we compare the performance of the policies for different values of system size ( $n$ ). For the results reported in Table 5.1, the “Empirical + Static Storage” policy learns for 0.1 s and the “Good Turing + Static Storage” policy learns for 0.7 s. The performance of all four policies improves as the system size increases and the adaptive policies significantly outperform the two learning-based static storage policies. Figure 5.4 is a plot of the mean values reported in Table 5.1.

In Table 5.2, we compare the performance of the policies for different values of Zipf parameter  $\beta$ . For the results reported in Table 5.2, the duration of the learning phase for both learning based policies is fixed such that the expected number of arrivals in that duration is 100. The performance of all four policies improves as the value of the Zipf parameter  $\beta$  increases, however, the MYOPIC policy outperforms both learning-based static storage policies for all values of  $\beta$  considered.

In Table 5.3, we compare the performance of the policies for different values of load  $\bar{\lambda}$ . For the results reported in Table 5.3, the duration of the learning phase for both learning based policies is fixed such that the expected number of arrivals in that duration is 100. The performance of all four policies deteriorates as the load increases, however, for all loads considered, the MYOPIC policies outperforms the two learning-based static storage policies.

| Policy                       | $n$  | Mean   | $\sigma$ |
|------------------------------|------|--------|----------|
| GENIE                        | 200  | 0.9577 | 0.0081   |
|                              | 400  | 0.9698 | 0.0045   |
|                              | 600  | 0.9752 | 0.0034   |
|                              | 800  | 0.9788 | 0.0030   |
|                              | 1000 | 0.9814 | 0.0025   |
| MYOPIC                       | 200  | 0.8995 | 0.0258   |
|                              | 400  | 0.9260 | 0.0167   |
|                              | 600  | 0.9380 | 0.0132   |
|                              | 800  | 0.9481 | 0.0101   |
|                              | 1000 | 0.9532 | 0.0080   |
| Empirical + Static Storage   | 200  | 0.6292 | 0.0662   |
|                              | 400  | 0.6918 | 0.0443   |
|                              | 600  | 0.7246 | 0.0353   |
|                              | 800  | 0.7464 | 0.0304   |
|                              | 1000 | 0.7622 | 0.0268   |
| Good Turing + Static Storage | 200  | 0.6875 | 0.0274   |
|                              | 400  | 0.7249 | 0.0180   |
|                              | 600  | 0.7443 | 0.0140   |
|                              | 800  | 0.7566 | 0.0118   |
|                              | 1000 | 0.7651 | 0.0104   |

Table 5.1: The performance of the four policies as a function of the system size ( $n$ ) for fixed values of load  $\bar{\lambda} = 0.8$  and  $\beta = 1.5$ . The values reported are the mean and standard deviation ( $\sigma$ ) of the fraction of jobs served. Both adaptive policies (GENIE and MYOPIC) significantly outperform the two learning-based static storage policies.

| Policy                       | $\beta$ | Mean   | $\sigma$ |
|------------------------------|---------|--------|----------|
| GENIE                        | 2       | 0.9939 | 0.0026   |
|                              | 3       | 0.9996 | 0.0015   |
|                              | 4       | 0.9998 | 0.0011   |
|                              | 5       | 0.9998 | 0.0012   |
|                              | 6       | 0.9998 | 0.0011   |
| MYOPIC                       | 2       | 0.9778 | 0.0078   |
|                              | 3       | 0.9960 | 0.0033   |
|                              | 4       | 0.9982 | 0.0026   |
|                              | 5       | 0.9990 | 0.0018   |
|                              | 6       | 0.9993 | 0.0013   |
| Empirical + Static Storage   | 2       | 0.8594 | 0.0194   |
|                              | 3       | 0.9228 | 0.0155   |
|                              | 4       | 0.9397 | 0.0119   |
|                              | 5       | 0.9453 | 0.0095   |
|                              | 6       | 0.9495 | 0.0073   |
| Good Turing + Static Storage | 2       | 0.8436 | 0.0235   |
|                              | 3       | 0.9198 | 0.0154   |
|                              | 4       | 0.9378 | 0.0124   |
|                              | 5       | 0.9456 | 0.0094   |
|                              | 6       | 0.9491 | 0.0072   |

Table 5.2: The performance of the four policies as a function of the Zipf parameter ( $\beta$ ) for fixed values of system size  $n = 500$  and load  $\bar{\lambda} = 0.9$ . The values reported are the mean and standard deviation ( $\sigma$ ) of the fraction of jobs served. The MYOPIC policy outperforms the two learning-based static storage policies for all values of  $\beta$  considered.



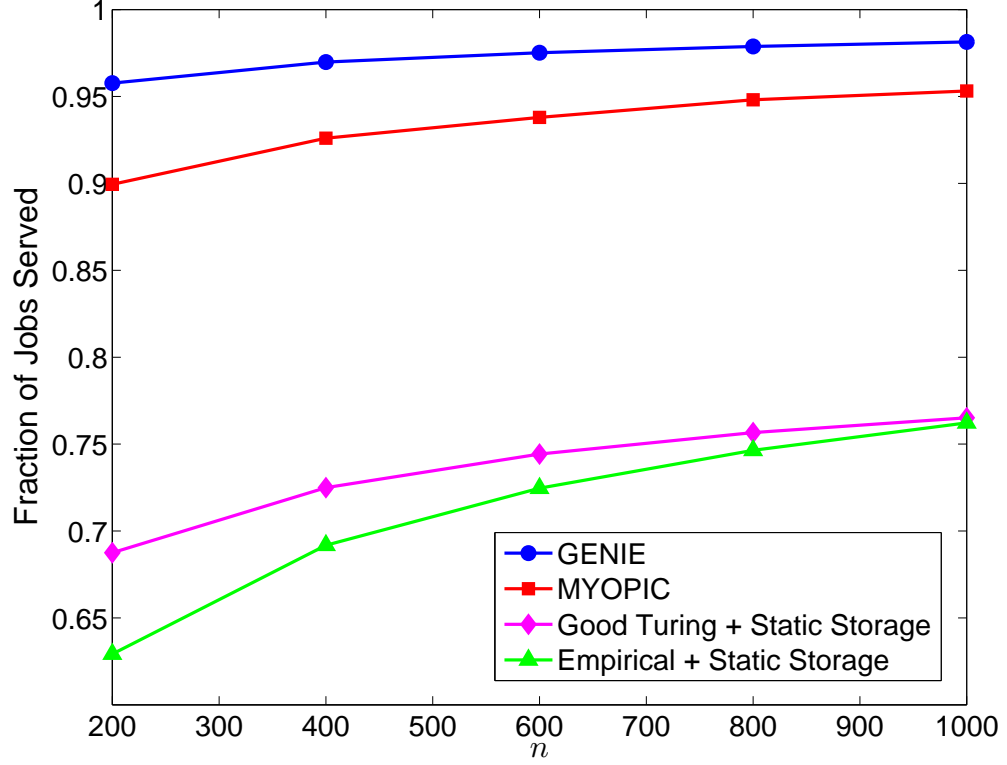


Figure 5.4: Plot of the mean values reported in Table 5.1 – performance of the storage policies as a function of system size ( $n$ ) for  $\bar{\lambda} = 0.8$  and  $\beta = 1.5$ .

In Figure 5.5, we plot the mean value (with error bars of  $3 \times \text{std. dev.}$ ) of the number of external fetches made by the MYOPIC and GENIE storage policies for different values of  $n$  and  $\beta$  for a load of 0.9 averaged over 10000 iterations. As expected, the GENIE storage policy makes more external fetches than the MYOPIC policy.

| Policy                       | $\bar{\lambda}$ | Mean   | $\sigma$ |
|------------------------------|-----------------|--------|----------|
| GENIE                        | 0.500           | 0.9892 | 0.0025   |
|                              | 0.725           | 0.9788 | 0.0013   |
|                              | 0.950           | 0.9531 | 0.0017   |
| MYOPIC                       | 0.500           | 0.9605 | 0.0113   |
|                              | 0.725           | 0.9484 | 0.0105   |
|                              | 0.950           | 0.8973 | 0.0221   |
| Empirical + Static Storage   | 0.500           | 0.7756 | 0.0222   |
|                              | 0.725           | 0.7705 | 0.0238   |
|                              | 0.950           | 0.7352 | 0.0235   |
| Good Turing + Static Storage | 0.500           | 0.7849 | 0.0230   |
|                              | 0.725           | 0.7589 | 0.0249   |
|                              | 0.950           | 0.6869 | 0.0348   |

Table 5.3: The performance of the four policies as a function of the load ( $\bar{\lambda}$ ) for fixed values of system size  $n = 500$  and  $\beta = 1.2$ . The values reported are the mean and standard deviation ( $\sigma$ ) of the fraction of jobs served. The MYOPIC policy significantly outperforms the two learning-based static storage policies for all loads considered.

## 5.5 Related Work

Our model of content delivery systems shares several features with recent models and analyses for content placement and request scheduling in multi-server queueing systems [58, 93, 59, 99]. All these works either assume known demand statistics, or a low-dimensional regime (thus permitting “easy” learning). Our study is different in its focus on unknown, high-dimensional and time-varying demand statistics, thus making it difficult to consistently estimate statistics. Our setting also shares some aspects of estimating large alphabet distributions with only limited samples, with early contributions from Good and Turing [38], to recent variants of such estimators [66, 94].

Our work is also related to the rich body of work on the content replication strategies in peer-to-peer networks, e.g., [89, 52, 65, 51, 105, 112, 23, 111]. Replication is used in various contexts: [89] utilizes it in a setting with large storage limits, [52, 65] use it to decrease the time taken to locate specific content, and [112, 23, 111] use it to increase bandwidth in the setting of video streaming. However, the common assumption is that the number of content-types does not scale with the number of peers, and that a request can be served in parallel by multiple servers (and with increased network bandwidth as the number of peers with a specific content-type increases) which is fundamentally different from our setting.

Finally, our work is also related to the vast literature on content replacement algorithms in server/web cache management. As discussed in [100], parameters of the content (e.g., how large is the content, when was it last requested) are used to derive a cost, which in-turn, is used to replace content. Examples of algorithms that have a cost-based interpretation include the Least Recently Used (LRU) policy, the Least Frequently Used (LFU) policy, and the Max-Size policy [101]. We refer to [100] for a survey of web caching schemes. There is a huge amount of work on the performance of replication strategies in single-cache systems; however the analysis of adaptive caching schemes in distributed cache systems under stochastic models of arrivals and departures is very limited.

## 5.6 Conclusions

In this chapter, we considered the high dimensional setting where the number of servers, the number of content-types, and the number of requests to be served over any time interval all scale as  $O(n)$ ; further the demand statistics are not known a-priori. This setting is motivated by the enormity of the contents and their time-varying popularity which prevent the consistent estimation of demands.

The main message of this work is that in such settings, separating the estimation of demands and the subsequent use of the estimations to design optimal content placement policies (“learn-and-optimize” approach) is order-wise suboptimal. This is in contrast to the low dimensional setting, where the existence of a constant bound on the number of content-types allows asymptotic optimality of a learn-and-optimize approach.

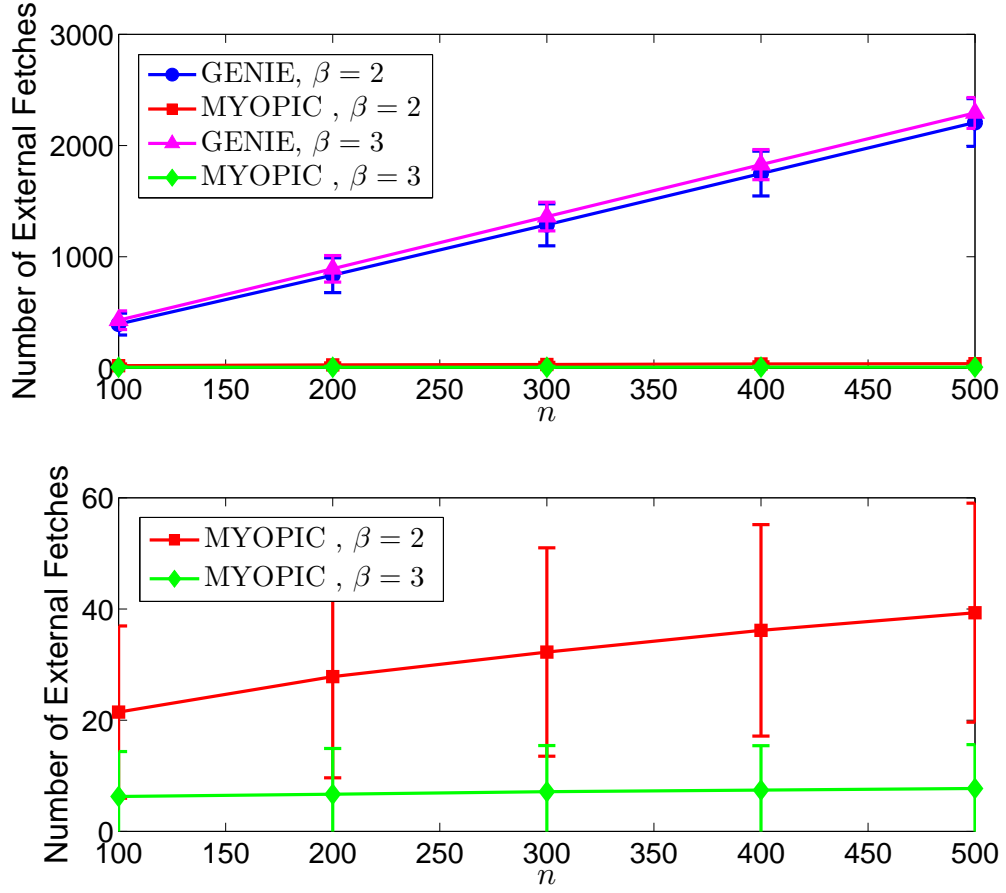


Figure 5.5: The mean number of external fetches (content fetched from the back-end server to place on a front-end server) by the two adaptive policies as a function of system size ( $n$ ) for  $\bar{\lambda} = 0.9$  and  $\beta = 2$  and 3. The first plot shows the performance of both GENIE and MYOPIC. The second plot focuses only on the performance of the MYOPIC storage policy for clarity.

## Chapter 6

# On Adaptive Content Replication in Large-Scale Content Delivery Networks

### 6.1 Introduction

Content Delivery Networks (CDNs) account for ever increasing fractions of network traffic (predicted to be more than 50% by 2018 [22]). At a high level, a CDN architecture is simple – it consists of *(i)* one or more “back-end” servers (or a cloud of servers) that have the entire content catalog, *(ii)* a large collection of front-end servers, each having a limited subset of the content, and *(iii)* a “director/scheduler” that matches Internet user content requests to appropriate front-end servers. The front-end servers each have limited capacity and storage – however, they can serve user content requests at a lower cost (compared to the back-end servers) due to their geographic/network proximity to the users. The key motivation behind using such network architectures is that serving requests through the front-end servers will effectively mitigate the traffic load on the network backbone, thus reducing the network bandwidth consumption. Naturally the content replication strategy (i.e., how many copies of each content should be stored on the front-end servers) forms an important part of the architecture.

Such content caching/replication systems have a rich literature (see Section 6.1.2). However, as noted in Chapter 5, recent usage patterns indicate that two important aspects need to be factored into algorithm design and performance analysis:

- (i) A large increase in *both* the number of content requests and content-types, and with new content being constantly added to the catalog (e.g. YouTube draws around a billion viewers per month, and 100 hours of new video uploaded per minute [109]).
- (ii) These videos are nowhere close to being equally popular. Indeed a wealth of studies in literature [17, 110, 43, 95] suggest that the popularity is heavy-tailed (Zipf’s law/distribution [114]), and that it changes often with time [34, 92].

Traditionally, the Independent Reference Model (IRM) has been used for modeling content request rates in CDNs [24, 34, 17, 110, 43, 95]. Under IRM, the content catalog has a fixed set of contents, and requests for various contents in the catalog arrive according to a generalized Zipf’s law. However, as noted in [92], IRM is not always suitable to model content popularity in CDNs, as it fails to capture the dynamics of the content catalog as well as the temporal changes in content popularity. To deal with the shortcoming of IRM, we use the Shot-Noise Model (SNM), proposed in [92] and validated using real traffic traces in [92] and [77]. Under SNM, each content is characterized by

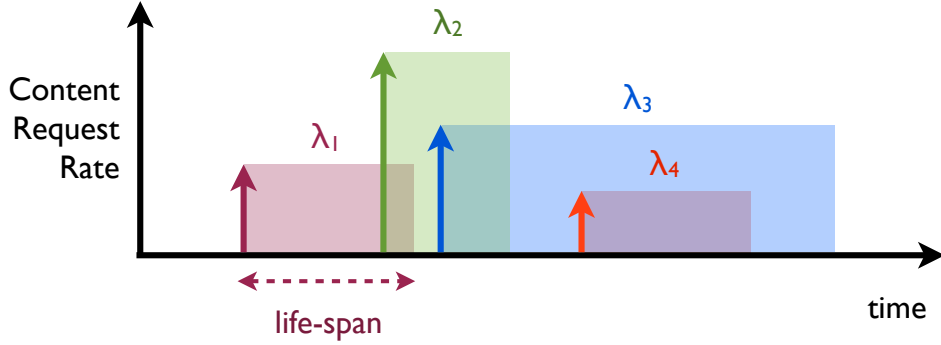


Figure 6.1: The SNM model [92] for the dynamics of content catalog and content popularity (arrows show the arrival of new contents to the catalog).

three parameters: (1) the time of arrival of the content into the catalog, (2) its life-span which is the length of the time interval after which the demand for the content effectively disappears, and (3) its popularity during its life-span (see Figure 6.1) according to a Zipf-like distribution.

The CDN setting we consider – large volumes of content, large volumes of content-types and time-varying popularity – is termed the *high-dimensional setting* in Chapter 5 because this setting does not permit a “good estimation” of demand statistics. This is unlike a traditional (low-dimensional) multi-server queueing system, where a small fraction of time can be allocated for learning demand or popularity statistics, and the rest of the time can be used for content delivery based on static optimization of content placement. The model we consider here is similar to that in Chapter 5 – it consists of a back-end server which stores the entire content catalog, which is assisted by  $n$  front-end servers with limited storage and service capabilities. Catalog contents can



be replicated on front-end servers by fetching them either from the back-end server or from the other front-end servers if possible. For instance, if an incoming request cannot be served by the front-end servers because the requested content-type is not available on any idle front-end server, the requested content can be fetched and replicated on an idle front-end server which can then serve the request.

However, two crucial differences distinguish this work from Chapter 5. First, the CDN incurs a cost for each content fetch. In our model, the cost of fetches from the back-end server is typically much higher than local fetches among the front-end servers since a fetch from the back-end server increases the load on the network backbone. Second, we use a more realistic dynamic content update model (the SNM model [92]) as opposed to an IRM-like model used in Chapter 5. As we will see, these more realistic features have important analytical and qualitative implications. Analytically, we convert the continuous time shot noise model (SNM) to a sequence of random-length blocks of time, where each block of time has the property that  $k$ -most popular files remain unchanged (for an appropriate choice of  $k$ ). Our analysis stitches the ‘local-analysis’ within each block over time. Qualitatively, we are now able to show, not just the gains due to adaptation, but indeed prove asymptotic optimality in the presence of fetching costs.

### 6.1.1 Contributions

Our main contributions are:

- We study a simple content replication policy – Least Recently Used with Replication (LRU-R) – a variant of the popular LRU algorithm that maintains multiple copies (replicates) of popular content on front-end servers. We show that for both IRM and SNM, this simple strategy is asymptotically optimal in the sense that it minimizes the total replication cost as the system size  $n \rightarrow \infty$ , thus showing that the LRU-like adaptive policy is robust to the dynamic changes in popularity and content catalog.
- Next we compare LRU-R with strategies that first estimate content (relative) popularities, and then use this information to statically optimize replication (i.e., how many copies of each content type). We show that such static strategies are order-wise suboptimal. Specifically, even though the adaptive policy comes with an associated cost per each content adaptation (unlike Chapter 5 with “free” adaptation), LRU-R still has a strictly lower overall cost.
- Finally, using simulations, we show that LRU-R indeed outperforms static-storage policies (as indeed the theory predicts). More interestingly, with the shot-noise model for content updates and arrivals, the LRU-R policy outperforms other popular variants such as the LFU (Least Frequently Used) algorithm. This is because a critical problem in caching/replication is to detect when a content type becomes unpopular (i.e., demand drops off) and hence, stop making copies and flush it from front-end servers. The LRU-R algorithm shines in this, and detects the end of

the life-span of a content faster than a LFU-like policy (the fast detection of the end of life-span is also important in the asymptotic optimality proofs for LRU-R).

### 6.1.2 Related Work

Most related to our work is Chapter 5 which studies replication strategies that maximize the number of requests served by front-end servers under IRM (fixed catalog) [24]. However Chapter 5 ignores the replication cost, i.e., assumes that frequent updates in the storage of servers can be made for “free”. The cost-based model considered in this chapter, in conjunction with the dynamic catalog (SNM), is a more practical model of CDNs.

Related content replication problems have been studied in the queuing literature [58, 93, 59, 99]; the key difference is that these studies implicitly assume that the popularity statistics can be learned easily (either known a-priori or the number of content types is fixed, and hence can be easily learned). Our setting of time-varying and scaling catalog size distinguish from this literature. Specifically, the high-level intuition in traditional settings – popularity statistics can be ‘easily’ learned by observing demand requests – no longer holds in our case.

Two other areas of work we refer to are those of web-caching [100] and peer-to-peer networks [89, 52, 65, 51, 105, 112, 23, 111]. However, the peer-to-peer models and issues are different from our setting (most importantly the bandwidth multiplier that arises from multiple peers serving the same

content). Further, as before, our scaling setting is fundamentally different (number of content types scaling up with load) from the web-caching and peer-to-peer literature. Next in the VoD setting, [9] proposes an optimization-based approach (learn the popularity and use a Mixed Integer Program for replication) that captures various costs (e.g., storage, bandwidth, placement) in detail.

Finally, the SNM model [92] has been used recently in [77] and [6] to study the performance of LRU for a single-cache system under non-stationary traffic. The stochastic models and analysis of request arrival and service considered in our work for the multi-server CDN are quite different from single-cache systems.

### 6.1.3 Basic Notation

$\text{Exp}(x)$  denotes an exponential random variable with parameter  $x$ .  $\text{Poisson}(x)$  denotes a Poisson random variable with parameter  $x$ . Given two real-valued random variables  $A$  and  $B$ ,  $A \leq_{st} B$  indicates stochastic dominance, i.e.,  $\mathbb{P}(A \geq x) \leq \mathbb{P}(B \geq x)$  for all  $x$ . For any functions  $f(\cdot)$  and  $g(\cdot)$ , we define  $\{f = O(g), f = o(g)\}$  if the limit  $\limsup_{n \rightarrow \infty} |f(n)/g(n)|$  is  $\{< \infty, = 0\}$ . Similarly, we let  $\{f = \Omega(g), f = \omega(g)\}$  if the relation  $g = \{O(f), o(f)\}$  is satisfied. When  $f = O(g)$  and  $f = \Omega(g)$ , we say that  $f = \Theta(g)$ . Finally, we use *w.h.p.* to indicate ‘with high probability’ as  $n \rightarrow \infty$ .

## 6.2 Setting and Model

In this section, we describe the model for content requests, storage, service, and the associated costs.

### 6.2.1 Catalog Dynamics

We consider the setting where contents arrive (e.g. are uploaded on Youtube) according to a Poisson process. As discussed in the introduction, in our setting, each content has a life-span, i.e., the interval after its arrival during which there is non-zero demand for the video. Once the life-span of a video elapses, the video is no longer requested by the users, i.e., the demand for the content disappears. This model has been validated for CDNs in [77] using trace-based simulations using data from the Orange network. Formally, we make the following assumptions on the catalog dynamics.

*Assumptions* (Catalog Arrival Process). a) The life-span of a content is exponentially distributed with mean  $\frac{1}{\gamma_1}n^c$ , where  $c \geq 0$  and  $\gamma_1 > 0$  are arbitrary constants.

b) Contents arrive (added to the catalog) according to a Poisson process with rate  $\gamma_2 n^{1-c}$ , where  $\gamma_2 > 0$  is some arbitrary constant.

Note that for any value of exponent  $c \geq 0$ , the expected number of active contents (i.e., content whose life-span has not elapsed) in the system at each time is  $O(n)$ . The exponent  $c$  captures how fast the set of active contents of the catalog changes, with smaller values of  $c$  corresponding to faster changes.

### 6.2.2 Arrivals and Content Requests

Content demand in CDNs has been observed to be heavy-tailed (Zipf's law) [34, 17, 110, 43, 95]. In addition, it has been observed in [4] that the large number of contents can be classified into a relatively small number of classes where all contents in a class have similar demand characteristics. We consider the following setting to capture these characteristics.

We assume that the contents in the catalog belong to a set of classes  $\mathcal{J}$ , with  $n^\alpha$  classes for some non-negative constant  $\alpha < 1$ . Each class  $i \in \mathcal{J}$  is characterized by a unique request arrival rate  $\lambda_i$ . Further the classes are indexed according to their order of popularity, i.e.,  $\lambda_1 > \lambda_2 > \dots > \lambda_{n^\alpha}$ .

Each arriving catalog content picks one class from  $\mathcal{J}$  independently, uniformly at random. Each content in class  $i$  has a request arrival rate of  $\lambda_i$  during its life-span.

*Assumptions* (Request Arrival Processes). a) Requests for content  $j$  in class  $i \in \mathcal{J}$  arrive as a Poisson process with rate  $\lambda_i$ . The class rates are described by the Zipf law with some parameter  $\beta > 2$ , i.e.,  $\lambda_i \propto 1/i^\beta$ ,  $i \in \mathcal{J}$ .

b)  $\sum_{i \in \mathcal{J}} \lambda_i = \frac{\gamma_1}{\gamma_2} \bar{\lambda} n^\alpha$  for some number  $\bar{\lambda} \in (0, 1)$ .

*Remark 3.* Under Assumptions 6.2.1 and 6.2.2, the total expected request arrival rate at any time will be  $\bar{\lambda}n$ . Hence  $\bar{\lambda}$  captures the effective load on the system.

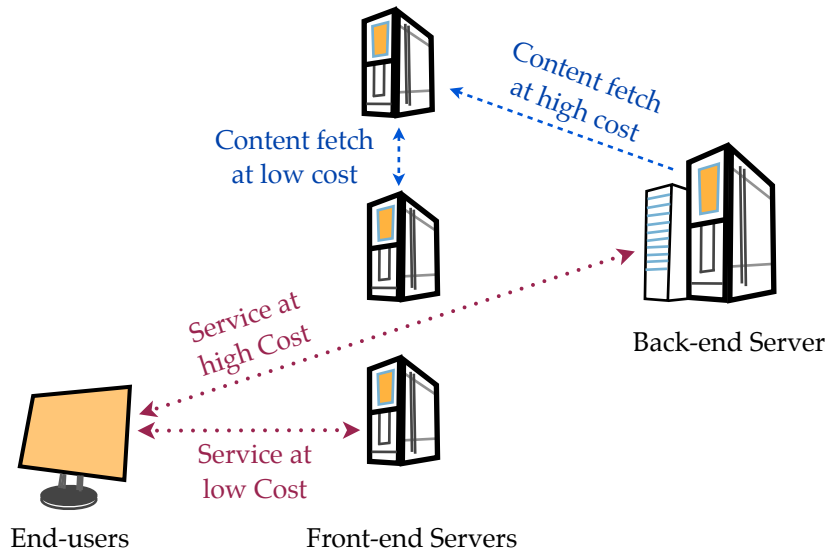


Figure 6.2: An illustration of a CDN with a back-end server and three front-end servers. User requests can be served both by the front-end servers (at low cost) and the back-end server (at high cost). Content can be replicated on the front-end servers by fetching it either from other front-end servers (at low cost) or from the back-end server (at high cost).

### 6.2.3 Server and Storage

We use the same server and storage model as in Chapter 5. The CDN (see Figure 6.2) consists of a back-end server which stores the entire catalog of contents and  $n$  front-end servers which can store one content piece each. Each front-end server can serve only one request at a time. The key difference is the *cost* model, wherein we now have a cost for each fetch (see the next section).

### 6.2.4 Service and Content Fetching

The service time is assumed to be  $\text{Exp}(1)$  (i.e., exponentially distributed, with mean time normalized to unity). When a request for a content arrives, it can be served in three ways:

- (i) By an idle front-end server with the specific content-type requested;
- (ii) By an idle front-end server that does not have the specific content-type. In this case, the specific content is fetched either from another front-end server or the back-end server (if no front-end server has this content), placed on the idle front-end server which then serves the request. The cost of fetching a content from the back-end server and a front-end server is  $C_b$  and  $C_f$  respectively, with  $C_f \leq C_b$ ;
- (iii) By the back-end server, which serves it at the highest cost  $C_m$ , with  $C_f \leq C_b \leq C_m$ . We assume that the back-end server has sufficient capacity to serve all the request that are routed to it.

## 6.3 Main Results and Discussion

We present and discuss the main results.

### 6.3.1 Static Catalog (IRM)

Before discussing our results for SNM where the content catalog is both large as well as time-varying (due to arrival of new content and time-varying content popularity), we provide intuition by first considering IRM where the catalog is large but static, i.e., the set of contents in the catalog as well as their



popularity does not change with time. This is also closely related to the Block Change Model (but with additional fetching costs) in Chapter 5 from which we borrow the notation below. Specifically we make the following assumptions:

*Assumptions* (Static Catalog). a) The content catalog has  $n$  contents.

b) Requests for each content  $j$  arrive according to a Poisson process with rate  $\lambda_j$  and  $\{\lambda_j\}$  follows the Zipf's law with parameter  $\beta > 2$ , i.e.,

$$\lambda_j = \bar{\lambda} n \frac{j^{-\beta}}{z(\beta)}, \text{ where } z(\beta) = \sum_{j=1}^n j^{-\beta} \text{ and } \bar{\lambda} \in (0, 1).$$

d) The service time for each request is  $\text{Exp}(1)$ .

e) The request rates remain unchanged for a time interval of length  $T(n)$ .

**Definition 1.** Given a time interval  $[0, T(n)]$ , let

- $U(T(n))$  be the number of unique contents requested in  $[0, T(n)]$ ,
- $X_j(t)$  be the number of requests for the content  $j$  being served at time  $t \in [0, T(n)]$ ,
- $Z_j(T(n)) = \max_{t \in [0, T(n)]} X_j(t)$ ,
- $Z(T(n)) = \sum_{j=1}^n Z_j(T(n))$ ,
- $C_\Psi(T(n))$  be the cost of the content replication policy  $\Psi$  to serve the requests over the time interval  $[0, T(n)]$ .

Our first theorem provides a lower bound on the cost of the optimal content replication policy (OPT) which knows the entire sample path, i.e., it knows the arrival sequence as well as service times of all the future requests.

**Theorem 23.** *Given a time interval  $[0, T(n)]$ , the cost of the optimal replication policy OPT is at least*

$$C_{OPT}(T(n)) \geq C_f Z(T(n)) + (C_b - C_f)U(T(n)).$$

Note that this result is not asymptotic and holds for any value of  $n$  and  $T(n)$ .

Next, we study a variant of the Least Recently Used (LRU) algorithm – an algorithm (with many variants) having a long history [102, 100]. These algorithms typically have been for cache-hit-ratio maximization in a setting with finite storage and unlimited service capacity. Our adaptive policy – LRU-Replicate (LRU-R) – described in Figure 6.3, replicates content on multiple servers. Thus, LRU-R can be interpreted as an extension of LRU to a multi-server system, each with limited storage and service capacity.

**Theorem 24.** *Under Assumption 6.3.1, if  $T(n) = O(n)$ , starting from an empty system, given any  $\delta \in (0, 1 - \bar{\lambda})$ , there exists a large enough  $N_\delta$  such that for all  $n \geq N_\delta$ ,*

- $C_{LRU-R}(T(n)) = C_{OPT}(T(n))$ ,
- $C_{LRU-R}(T(n)) \leq C_f(1 - \delta)n + (C_b - C_f)U(T(n))$ ,

---

```

1: When a request for content  $j$  arrives do,
2:   if no idle front-end server, then
3:     forward request to back-end server.
4:   else
5:     if an idle server has content  $j$ , then
6:       forward request to that server.
7:     else
8:       fetch content  $j$  locally from a front-end server storing content  $j$  if
       available; otherwise fetch it from back-end server.
9:       among idle servers, replace the content that was least recently re-
       quested with content  $j$  and serve request.
10:    end if
11:  end if

```

---

Figure 6.3: LRU-R – An LRU variant that replicates content among several front-end servers.

*with probability greater than  $1 - \frac{1}{n}$ .*

We thus conclude that as the system size ( $n$ ) goes to infinity, the performance of the LRU-R policy is the same as that of the optimal storage policy (OPT) with high probability. This result is surprising because the LRU-R policy is a simple adaptive (and online) policy which has no knowledge of content popularity (or the sample path) and yet in the high-dimensional setting considered here, it has the same performance as the optimal policy which knows the entire sample path.

Next, we characterize the performance of learning-based static storage policies. This class, introduced in Chapter 5, serves as a benchmark to compare with adaptive policies. In a low-dimensional setting (where the number of content types do not scale), it is easy to argue that static policies incur

vanishingly small cost for learning popularities (basically, the learning cost is amortized over time); thus, static policies have traditionally been used in many queueing settings for characterizing stabilizing policies. A static policy has two parts – a learning part and a storage/optimization part. Given an interval of time  $T(n)$ , the policy divides this into two sub-intervals – *Phase 1* and *Phase 2* (see Figure 6.4). In Phase 1, the policy empirically estimates popularities based on the request seen over this time sub-interval. At the beginning of Phase 2, the policy statically chooses which content and how many of each is loaded on the front-end servers, and this loading remains static over the rest of this time sub-interval. The choice of the lengths of the two sub-intervals can depend on system parameters such as  $T(n)$  and the Zipf distribution. In Section 6.4, we will consider different types of learning algorithms (specifically, those designed for a high-dimensional regime). Note that the algorithms studied in [58, 59, 9] are in this class.

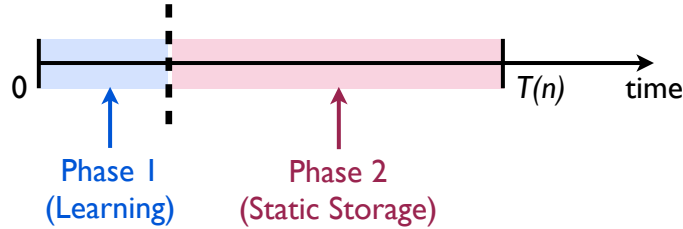


Figure 6.4: The time interval (denoted by  $T(n)$ ) is divided into two phases: Phase 1 – Learning, and Phase 2 – Storage/Optimization; figure adapted from Chapter 5.

The next theorem provides a lower bound on the cost of any learning-based static storage policy ( $C_{\text{Learning}}$ ) for our setting where unlike Chapter 5,

the CDN incurs a cost per content fetch.

**Theorem 25.** *Under Assumption 6.3.1, starting from an empty system,*

$$\mathbb{E}[C_{\text{Learning}}(T(n))] \geq \max(\min(C_m n, \Gamma_1(T(n))), \Gamma_2(T(n))),$$

where

$$\Gamma_1(T(n)) = \mathbb{E}[C_{\text{opt}}(T(n))] + (C_m - C_f)\Omega(nT(n))^{\frac{1}{2-1/\beta}},$$

$$\Gamma_2(T(n)) = C_m\Omega(nT(n))^{\frac{1}{2-1/\beta}}.$$

Next we compare the performance of LRU-R with the performance of learning-based static storage policies.

**Corollary 3.** *Let  $r = 1 - 1/\beta \in (1/2, 1)$ . Under Assumption 6.3.1, starting from an empty system,*

*Case 1:  $T(n) = \Omega(1)$  and  $T(n) = O(n^r)$ :*

$$\begin{aligned} \mathbb{E}[C_{\text{Learning}}(T(n))] &= \mathbb{E}[C_{\text{opt}}(T(n))] \\ &\quad + (C_m - C_f)\Omega(nT(n))^{\frac{1}{1+r}}, \\ C_{\text{LRU-R}}(T(n)) &= C_{\text{opt}}(T(n)), \text{ w.h.p.} \end{aligned}$$

*Case 2:  $T(n) = \omega(n^r)$  and  $T(n) = O(n)$ :*

$$\begin{aligned} \mathbb{E}[C_{\text{Learning}}(T(n))] &= \omega(n), \\ C_{\text{LRU-R}}(T(n)) &= C_{\text{opt}}(T(n)) = O(n), \text{ w.h.p.} \end{aligned}$$

Thus Corollary 3 indicates that learning-based static storage policies are sub-optimal. In particular, when  $T(n)$  scales faster than  $n^r, r \in (0.5, 1)$  (Case 2), such policies are order-wise suboptimal. In Chapter 5, where storage could be adapted for free (no content fetching cost), the sub-optimality of learning-based static storage policies could be attributed to the fact that adaptive policies have more flexibility than learning-based static storage policies at no extra cost. However, in our setting, even though content adaptation comes with an associated cost, learning-based static storage policies are still suboptimal.

We now shift our focus to the Shot Noise Model (SNM) where the catalog size increases with time.

### 6.3.2 Dynamic Catalog (SNM)

Recall the SNM model described in Section 6.2, where new content types are added to the catalog dynamically over time. In the static catalog (IRM model), the content popularity was fixed and the policy needed to keep enough copies of various contents on the front-end servers, whereas here the arrival of new content types and expiration of the life-spans of active contents can potentially change the entire popularity ordering of the active contents. In this setting, the contents stored on the front-end servers need to be modified when new popular contents arrive into the catalog in order to serve the requests; however, the learning-based static storage does not allow such modifications, thus yielding an even worse cost than the one under IRM. Again our

goal is to design an efficient replication policy with minimum adaptation cost. Motivated by the fact that the LRU-R policy performed well for the static catalog, we continue to evaluate the performance of the LRU-R policy for the dynamic setting of SNM.

**Theorem 26.** *Under Assumptions 6.2.1 and 6.2.2, if  $3\alpha - 2 + 2c > 0$ ,  $\alpha(\beta - 1) > 0$  and  $T(n) = o(n^\epsilon)$  where  $\epsilon = \min \left\{ \frac{\beta-1}{2\beta+1}(3\alpha - 2 + 2c), \alpha(\beta - 1) \right\}$ , starting from an empty system,*

$$\frac{C_{LRU-R}(T(n))}{C_{OPT}(T(n))} \rightarrow 1, \text{ w.h.p. as } n \rightarrow \infty.$$

It is easy to see from the SNM model that the time-scale of changes in the popularity (as a result of expiration of a life-span or arrival of a new content) is  $O(n^{c-1})$ . To make a connection between the results in this section, let  $c \leq 2$  which means measuring the cost over the same  $O(n)$  time-scale as in Theorem 24. Then, for any  $\beta > 2$ , by choosing  $\alpha$  close enough to 1, we can ensure  $c - 1 < \epsilon$  in Theorem 26. We thus conclude that the LRU-R content placement policy is asymptotically optimal (in the sense of cost ratio) even under the dynamic setting of SNM.

## 6.4 Simulation Results

In this section, we present simulation results for IRM as well as SNM.

### 6.4.1 Static Catalog (IRM)

We compare the performance of the LRU-R policy and the following two learning-based static content replication policies proposed in Chapter 5:

1. *Empirical + Static Storage*: This is the standard empirical estimator – the number of copies of a content type is directly proportional to the number of requests for that content type in Phase 1.
2. *Good Turing + Static Storage*: The Good-Turing estimator [38, 30, 66] is used to estimate both the popularities of content types seen during Phase 1 as well as those of content types *not seen* in Phase 1. Thus, this estimator tries to account for the fact that in a high-dimensional setting, there are several content-types that are simply not even seen during Phase 1. Further, the estimator assigns all the content types that have not been seen to have equal popularity; however, these are assigned a smaller popularity than content types that have been seen. The key algorithmic operation in the Good-Turing estimator is to determine the relative popularities assigned to those that have been seen vs. those that are unseen. Please refer to Section IV in Chapter 5 for a detailed description of the policy, and [66, 94] for some theoretical guarantees for this type of policy.

We compare the performance of the policies for different values of system size ( $n$ ) and plot the mean cost over a duration of 5 s ( $T(n) = 5$ ), by averaging over 10000 simulations for each policy. The learning time for the



learning-based policies is 0.1 s, the Zipf parameter  $\beta = 2$  and the load  $\bar{\lambda} = 0.9$ . Recall that our setting has three types of costs:  $C_f$  (the cost of fetching a content from a front-end server to replicate on another front-end server),  $C_b$  (the cost of fetching a content from the back-end server to replicate on another front-end server) and  $C_m$  (the cost of serving a request using the back-end server), such that  $C_f \leq C_b \leq C_m$ . Without loss of generality, we fix the value of  $C_f$  to 1 and present simulation results for two sets of values for  $C_b$  and  $C_m$ . For the first plot (Figure 6.5), we assume that  $C_f = C_b = C_m = 1$  and for the second plot (Figure 6.6),  $C_f = 1$ ,  $C_b = C_m = 10$ . We see that LRU-R significantly outperforms the two learning-based static storage policies in both cases and the performance gap increases for higher values of  $C_b$  and  $C_m$ .

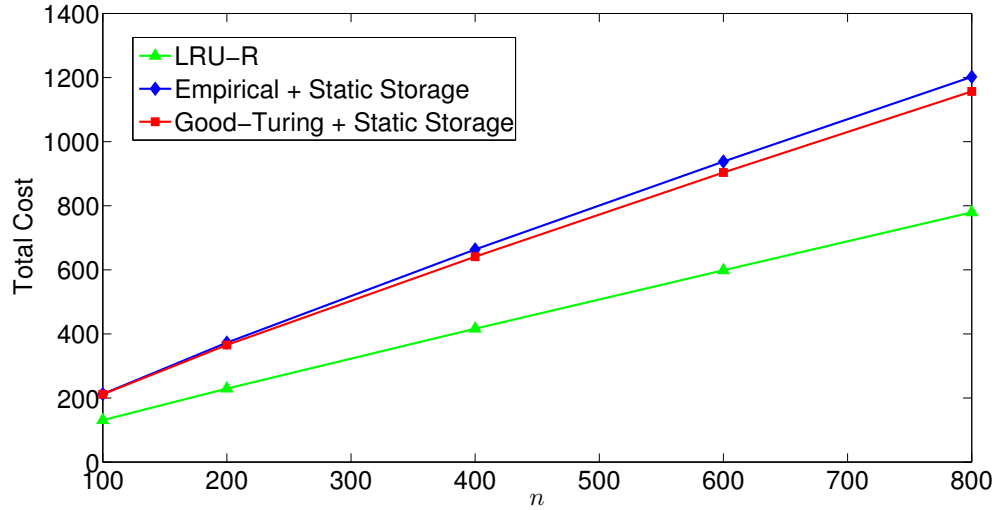


Figure 6.5: The cost of content replication policies as a function of system size ( $n$ ) for  $\bar{\lambda} = 0.9$ ,  $\beta = 2$ ,  $C_f = C_b = C_m = 1$ .

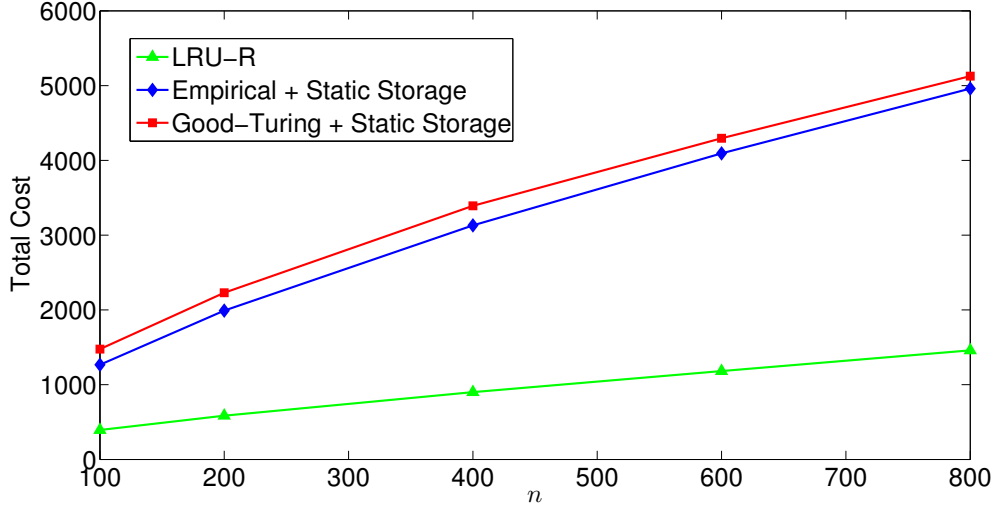


Figure 6.6: The cost of content replication policies as a function of system size ( $n$ ) for  $\bar{\lambda} = 0.9$ ,  $\beta = 2$ ,  $C_f = 1$ ,  $C_b = C_m = 10$ .

#### 6.4.2 Dynamic Catalog (SNM)

As mentioned in Section 6.3, under the SNM, adaptation is necessary because the contents stored on the front-end servers need to be modified as new contents arrive in order to serve the requests using the front-end servers. Here we investigate the effect of adaptation based on metrics other than LRU. In particular, we compare the performance of the LRU-R policy with a LFU-based adaptive policy called LFU-Replicate (LFU-R), that tries to replicate most frequently used contents on idle servers. Formally, for the LFU-R policy, in Step 9 in Figure 6.3, we replace “recently” by “frequently”. We compare the performance of the policies for different values of system size ( $n$ ) over a duration of 30 s ( $T(n) = 30$ ). For the results presented in Figure 6.7, the

Zipf parameter  $\beta = 4$ , the load  $\bar{\lambda} = 0.9$ ,  $\alpha = 0.5$ ,  $c = 0$ ,  $\gamma_1 = 1$ ,  $\gamma_2 = 0.33$ ,  $C_f = C_b = C_m = 1$ . We see that LRU-R significantly outperforms the LFU-R for large  $n$ . Hence, although adaptation can improve the performance, the choice of LRU based adaptation seems crucial to get asymptotic optimality.

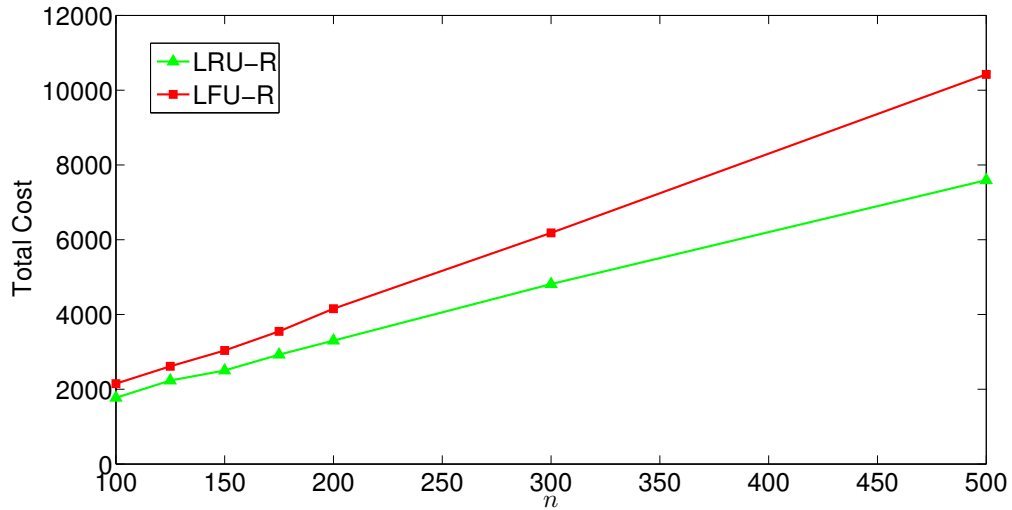


Figure 6.7: The cost of content replication policies as a function of system size ( $n$ ) for  $\bar{\lambda} = 0.9$ ,  $\beta = 4$ ,  $C_f = C_b = C_m = 1$ .

## 6.5 Conclusions

In this chapter, we studied content placement in large-scale content delivery networks, where content can be replicated/updated on the servers, but by incurring a cost per replicate/update. We showed that a simple adaptive policy called LRU-R, which makes online replication decisions based on an LRU metric, asymptotically minimizes the total cost over time. We proved

the result under both static and dynamic models, thus also demonstrating the robustness of LRU-R to temporal changes in catalog and content popularity. The simulation results indeed verify the theoretical results, and further suggest that the use of LRU metric, as opposed to other metrics like LFU, is necessary to get any asymptotic optimality for the adaptive policy.

## Chapter 7

### Conclusions

In this dissertation, we have presented and analyzed resource allocation algorithms for large-scale multi-server systems arising from two applications – multi-channel wireless communication networks and large-scale content delivery networks. For each resource allocation problem, we take the more appropriate modeling approach – stochastic or adversarial and use appropriate performance metrics (stability/queue-length performance/delay performance/goodput/competitive ratio). In spite of these difference, the large-scale of these problems motivates the need to design algorithms which are simple/greedy/distributed and thus scalable, yet, have rigorous performance guarantees.

One common lesson we draw from all the problems studied in this dissertation is that although the large-scale of these problems necessitates algorithms that are scalable, thus restricting the kind of techniques that can be employed by the resource allocation algorithms, it also provides flexibility due to the *degrees of freedom*, which if carefully utilized can ensure that simple/greedy/iterative and “sloppy” algorithms which have incomplete state information, have good performance. For instance,

- In general multi-hop settings, it is known that MaxWeight type policies are not stabilizing, however, in the large-scale instantiations we consider in Chapter 2, variants of the MaxWeight algorithm stabilize the system without the knowledge of downstream queue-length information. The key factor which facilitates this is the huge amount of diversity in the system as a result of the large number of frequency channels and relays. We exploit this diversity by proposing a simple iterative channel allocation procedure, where each channel decides which user to serve in a greedy manner using a cyclic tie-breaking policy. For this simple channel allocation procedure, we show that even if the routing algorithm makes “mistakes” in the first hop (base-station to relays), i.e., it forwards packets to a relay which is already backlogged, there is sufficient flexibility in the second hop (relays to users) due to the large number of OFDM frequency channels to ensure that the system is stabilized.
- In Chapter 3, the goal is to design a distributed scheduler which can stabilize the system without any coordination between the ANs. To achieve this, we propose the DIST scheduler, where each AN implementing the DIST scheduling policy decides which frequency channels to use in a greedy manner, breaking ties at random. The number of frequency channels being large ensures that the event that two nearby ANs pick the same channels for transmission is quite unlikely, thus the probability that transmissions of the two ANs interfere with each other is low. This ensures that the system is stabilized by the DIST scheduler even without

coordination between ANs.

- In Chapter 5, in the large-scale setting we consider, even without any information about content popularity, a simple greedy policy (MYOPIC) ensures that all but the first request for each content is served at a low cost by the network. The MYOPIC policy uses recently requested contents as proxies for the more popular contents and replicates recently requested contents on idle servers. The large number of servers in the system ensure that for any load which can be supported by the system, with high probability, the number of idle servers in the system at any time is large enough to ensure that for all popular contents, once the content type is requested for the first time, there is always at least one idle server which can serve an incoming request for that content.

We therefore conclude that for the resource allocation problems we consider, carefully designed low-complexity resource allocation algorithms which effectively exploit the flexibility and diversity in the system due to its large-scale can satisfy the dual purpose of good performance and scalability.

## Appendices



# Appendix A

## Proofs from Chapter 2

### A.1 Large System Stability of Iterative Max Weight

We consider the FD-w/oDL and HD-wDL models separately. We first provide a proof outline for the FD-w/oDL model.

#### A.1.1 FD-w/oDL

1. We first prove that if  $\lambda > S_{max}$ , no scheduling policy can stabilize the system.
2. We then show that the base-station queues are stable for any  $\lambda < S_{max}$ . This proof uses the fact that since there are  $R(n)$  relays, for large  $n$ , every channel can be used at  $S_{max}$  to send packets to at least one of these relays with very high probability. As  $\lambda < S_{max}$ , with high probability, fewer packets come into the system in a time slot than the number that can be served, thus ensuring that the base-station queues are stable.
3. Since the arrival process at the base-station queues is stationary and ergodic, and the base-station queues are stable, the arrival process at the relay queues (which is the departure process of the base-station queues) is also stationary and ergodic. By Theorem 5 in [15], we know that the

SSG algorithm is throughput optimal for the system consisting of just the relay queues. Therefore, to prove that the MaxWeight SSG algorithm stabilizes the relay queues, we need to show that the arrival process at the relays is inside the throughput region of the relays queues.

4. Since the throughput region of the relays queues is not known, to do this, we propose an algorithm called the Arrival Prioritized-SSG (AP-SSG) algorithm and show that this algorithm can stabilize the relay queues for the arrival process which is the departure process of the base-station queues. This shows that the departure process of the base-station queues lies in the throughput region of the relay queues and therefore, the relay queues will also be stabilized by the throughput optimal SSG algorithm.
5. The AP-SSG algorithm stores 2 values corresponding to each relay queue. Before allocation for slot  $t$  begins, the first value  $A_i^{r(0)}$  is initialized to the number of arrivals to that queue at the beginning of slot  $t$  and the second value  $R_{ri}^{(0)}$  is initialized to the queue length of the queue for user  $i$  at relay  $r$  at the end of time-slot  $t - 1$ .

The allocation proceeds in  $n$  rounds. In round  $k$ , the algorithm finds a queue with the highest  $A_i^{r(k-1)}X_{i,k}^r$  value. If this value is greater than 0, channel  $k$  is allocated to queue  $i$  at relay  $r$  and  $A_i^{r(k)}$  is updated to  $(A_i^{r(k-1)} - X_{i,k}^r)^+$ . If  $A_i^{r(k-1)}X_{i,k}^r = 0$ , the algorithm finds a queue with the highest  $R_{ri}^{(k-1)}X_{i,k}^r$  value and serves it. It updates  $R_{ri}^{(k)}$  to  $(R_{ri}^{(k-1)} - X_{i,k}^r)^+$ .

The AP-SSG algorithm therefore, gives the first priority to queues which

have packets that arrived at the beginning of that slot and then to queues which are the most backlogged. For the AP-SSG algorithm, we prove the following key lemma.

*Lemma 3.* Let  $S_{ri}$  be the service allocated to queue  $i$  at relay  $r$  by the AP-SSG algorithm. Let  $E_4$  be the event that

$$\cap_{r,i} \{A_i^r \leq S_{ri}\} \cap \{S_{r^*i^*} \geq A_{i^*}^{r^*} + S_{max}\},$$

where  $\{r^*, i^*\} \in \arg \max_{r,i} R_{ri}(t-1)$ . The event  $E_4$  implies that all the arrivals to the relay queues at the beginning of slot  $t$  are served in slot  $t$  and the at least one of the longest relay queues is served by at least 1 additional channel at  $S_{max}$ . Then, under Assumption 2,

$$P(E_4^c) = o\left(\frac{1}{n}\right).$$

The above lemma essentially shows a negative drift of at least  $R_{max}S_{max}$ , where  $R_{max}$  is the maximum queue length of the relay queues at the end of time-slot  $t-1$ . We then show that there exists an  $n_0$  such that this algorithm stabilizes the relay queue system with  $n > n_0$  channels via the quadratic lyapunov function. This proves that the arrival process at the relay queues which is the departure process of the base-station queues lies inside the throughput region of the relay queues and therefore, the relay queues will be stabilized by the SSG algorithm.

The following Lemma generalizes Theorem 4 in [16]. Theorem 4 in [16] was restricted to computing the stationary distribution of Markov Chains such

that in each time-slot, the value of the Markov random variable could increase by at most a constant number ( $k_0$ ) with exponentially small probability ( $e^{-cn}$ ). This lemma generalizes the theorem to markov chains which increase by at most  $\chi(n)$  in a given slot with probability at most  $f(n)$  such that  $\chi(n)^3 f(n) = o(1/n^2)$ .

*Lemma 4.* Consider a discrete time Markov Chain  $Y^{(n)} \in \{0, 1, 2, \dots\}$ . Let  $f(n) = o(\frac{1}{n^6})$  and  $\chi(n)$  such that  $\chi(n)^3 f(n) = o(1/n^2)$ . Suppose that the transition probabilities are as follows:

If  $Y^{(n)}(t) > 0$ ,

$$\begin{aligned} P(Y^{(n)}(t+1) = Y^{(n)}(t) - 1) &= 1/2 \\ P(Y^{(n)}(t+1) = Y^{(n)}(t) + \chi(n)) &= f(n) \\ P(Y^{(n)}(t+1) = Y^{(n)}(t)) &= 1/2 - f(n). \end{aligned}$$

If  $Y^{(n)}(t) = 0$ ,

$$\begin{aligned} P(Y^{(n)}(t+1) = \chi(n)) &= f(n) \\ P(Y^{(n)}(t+1) = 0) &= 1 - f(n). \end{aligned}$$

Let  $\pi(m) = P(Y^{(n)}(t) = m)$ . For this Markov Chain, we have that,

$$1 - \pi(0) \leq 4\chi(n)^3 f(n) = o\left(\frac{1}{n^2}\right).$$

*Proof:* Consider the Lyapunov function  $Lyap(x)=x$ . For  $n$  large enough, we have

$$\begin{aligned} E(Y^{(n)}(t+1) - Y^{(n)}(t) | Y^{(n)}(t), Y^{(n)}(t) > 0) \\ \leq \chi(n)f(n) - \frac{1}{2} \leq -\frac{1}{3}, \end{aligned}$$

so the Lyapunov function has negative drift outside the set  $\{0\}$  and therefore the Markov Chain is positive recurrent. The Markov Chain is also irreducible and aperiodic and therefore has a unique stationary distribution. We prove the following statement by induction about  $\pi(m)$  by induction

$$\pi(m) \leq \pi(0)(2\chi(n))^{2\lceil m/\chi(n) \rceil} f(n)^{\lceil m/\chi(n) \rceil}.$$

For  $n$  large enough,  $2\chi(n)^2 f(n) < 1$ .

Case I:  $1 \leq m \leq \chi(n)$

$$\begin{aligned} \pi(m) &= 2 \sum_{r=1}^m \pi(m-r) \sum_{j=r}^m f(n) \\ &\leq 2m^2 \pi(0) f(n) \\ &\leq 2(\chi(n))^2 \pi(0) f(n). \end{aligned}$$

Case II:  $(k-1)\chi(n) < m \leq k\chi(n)$

$$\begin{aligned} \pi(m) &= 2 \sum_{r=1}^{\chi(n)} \pi(m-r) \sum_{j=r}^{\chi(n)} f(n) \\ &\leq 2(\chi(n))^2 \pi(m-\chi(n)) f(n) \\ &\leq 2\chi(n)^2 \pi(0) 2^{k-1} (\chi(n))^{k-1} f(n)^{k-1} f(n) \\ &= (2\chi(n))^{2k} \pi(0) f(n)^k, \end{aligned}$$

thus completing the proof by induction.

Let  $n$  be large enough such that  $W = 2\chi(n)^3 f(n) < 1/2$ , then, by adding the values of  $\pi(m)$  for  $m = 0$  to  $\infty$  and equating it to 1, we get that,

$$\begin{aligned} 1 - \pi(0) &\leq \frac{W}{1 - W} \\ &\leq 2W \\ &= 4\chi(n)^3 f(n). \end{aligned}$$

■

In the following lemma, we prove that if on average, more than  $nS_{max}$  packets come into the system in every slot, no scheduling policy can stabilize the system.

*Lemma (1).* Under Assumption 2, if  $\frac{1}{n}E\left[\sum_{i=1}^n A_i(0)\right] = \lambda > S_{max}$ , then the system is unstable under any scheduling algorithm.

*Proof:* If  $\lambda > S_{max}$ , then the mean number packet arrivals to the system in a given time-slot is more than the maximum number of packets that can be served by the base-station or the relays in a given time-slot ( $= nS_{max}$ ). Hence the system is unstable under any scheduling algorithm.

■

We now prove that if  $\lambda < S_{max}$ , the SSG MaxWeight algorithm stabilizes the system. To handle coupled queues across hops (and the routing induced by

multiple hops and paths), our proof is iterative across hops. We first look at the base-station queues.

*Lemma 5.* Under Assumptions 2 and the SSG MaxWeight algorithm, given any arrival process such that  $\lambda < S_{max}$ , the markov chain  $(\mathbf{Q}(t), \mathbf{A}(t))$  is positive recurrent for  $n$  large enough.

*Proof:* We say that the base-station queue are stable if the SSG MaxWeight algorithm makes the base-station queues an aperiodic Markov Chain with a single communicating class which is positive recurrent.

Consider the Markov chain  $(\mathbf{Q}(t), \mathbf{A}(t))$  and the lyapunov function  $Q(t)$  where  $Q(t) = \sum_{i=1}^n Q_i(t)$ .

Consider the finite set  $F = \{\mathbf{Q} : \max_{1 \leq i \leq n} Q_i \leq nS_{max}\}$ . In this set,

$$\begin{aligned} & E[Q(t+1) - Q(t) | \mathbf{Q}(t), \mathbf{A}(t)] \\ = & E \left[ \sum_{i=1}^n Q_i(t+1) - \sum_{i=1}^n Q_i(t) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right] \\ \leq & n\lambda < \infty, \end{aligned}$$

by Assumption 2(a). Outside the set  $F$ ,

$$\begin{aligned}
& E[Q(t+1) - Q(t) | \mathbf{Q}(t), \mathbf{A}(t)] \\
= & E \left[ \sum_{i=1}^n Q_i(t+1) - \sum_{i=1}^n Q_i(t) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right] \\
= & E \left[ \sum_{i=1}^n \left( Q_i(t) + A_i(t+1) - \right. \right. \\
& \left. \left. \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \right)^+ - Q(t) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right] \\
\stackrel{(a)}{=} & E \left[ \sum_{i=1}^n A_i(t+1) \right. \\
& \left. - \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right] \\
= & E \left[ \sum_{i=1}^n A_i(t+1) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right] \\
& - E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right] \\
= & n\lambda - E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| \mathbf{Q}(t), \mathbf{A}(t) \right].
\end{aligned}$$

Where (a) follows from the fact that outside the set  $F$ , since  $\max_{1 \leq i \leq n} Q_i > nS_{max}$  the base station always has packets to send on all channels, therefore, no capacity is wasted. Let  $3\epsilon = \frac{S_{max}-\lambda}{S_{max}}$ . Consider the event  $E$  that there exists a set  $J$  of channels such that  $|J| = 2n\epsilon$  and  $X_{i,j}^{B,r} < S_{max}$  for all  $j \in J$  and



$$1 \leq r \leq R(n).$$

$$\begin{aligned} E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| E^c \right] &\geq (1-2\epsilon) S_{max} n, \\ E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| E \right] &\geq 0. \end{aligned}$$

Therefore,

$$\begin{aligned} &E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \right] \\ &= E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| E \right] P(E) \\ &\quad + E \left[ \sum_{j=1}^n X_{i,j}^{B,r}(t+1) Y_{i,j}^{B,r}(t+1) \middle| E^c \right] P(E^c) \\ &\geq (1-2\epsilon) S_{max} n P(E^c). \end{aligned}$$

By Assumption 2(b),  $P(E^c) = o\left(\frac{1}{n^6}\right)$  and therefore, for  $\lambda < S_{max}$  and  $n$  large enough,

$$\begin{aligned} &E[Q(t+1) - Q(t) | \mathbf{Q}(t), \mathbf{A}(t)] \\ &\leq n\lambda - (1-2\epsilon) S_{max} n P(E^c) \\ &\leq -1/2. \end{aligned}$$

Therefore, by Foster's theorem, the Markov Chain  $\mathbf{Q}(t)$  is positive recurrent. Now consider the Markov Chain  $Q(t)$ . We need to compute  $P(Q(t) > 0)$  to prove that the relay queues are stable. To this end, we study the Markov Chain  $Y^{(n)}(t)$  defined in Lemma 4 for  $f(n) = o(1/n^6)$  and  $\chi(n) = k_1 n^2$ . Note that by Theorem 3 in [16],  $Q(t) \leq_{st} Y^{(n)}(t)$  where  $Q(t) \leq_{st} Y^{(n)}(t) \Rightarrow P(Q(t) >$

$x) \leq P(Y^{(n)}(t) > x), \forall x$ . By Lemma 4 we have that, for  $n$  large enough, for the Markov Chain  $Y^{(n)}(t)$ ,

$$\begin{aligned} 1 - \pi(0) &\leq \frac{W}{1 - W} \\ &\leq 2W \\ &= 4(k_1 n^2)^2 P(E^c). \end{aligned}$$

Therefore,  $P(Q(t) > 0) \leq 4k_1^2 n^4 P(E^c) = o\left(\frac{1}{n^2}\right)$ .

■

We now look at the relay queues. We note that the departure process of the base-station queues is the arrival process of the relay queues. Since the arrival process of the base-station queues is stationary and ergodic and the base-station queue system is stable, the departure process is also stationary and ergodic and therefore, the arrival process of the relay queues is stationary and ergodic. Additionally, if we prove that the departure process of the base-station queues is inside the throughput region of the relay queues, then we have that the SSG algorithm will stabilize the relay queues. Since the SSG algorithm is throughput optimal for the system consisting of just the relay queues and users by Theorem 5 in [15].

To prove that the departure process of the base-station queues is inside the throughput region of the relay queues, we prove that there exists an algorithm that can stabilize the relay queues for the arrival process which is

the departure process of the base-station queues. We call this algorithm the Arrival Prioritized-SSG (AP-SSG) algorithm.

**Definition:** The AP-SSG algorithm allocates channels to queues in time-slot  $t$  according to the following procedure.

**Input:**

1. The queue lengths  $R_{ri}(t-1)$ , for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ .
2. The arrival vector  $A_i^r(t)$ , for for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ .
3. The channel realizations  $X_{ij}^r(t)$ , for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ ,  $1 \leq j \leq n$ .

**Steps:**

1. Initialize  $k = 1$  and  $Y_{ij}^r(t) = 0$ ,  $R_{ri}^{(0)}(t) = R_{ri}(t)$ ,  $A_i^{r(0)}(t) = A_i^r(t)$  for  $1 \leq i \leq n$ ,  $1 \leq r \leq R(n)$ ,  $1 \leq j \leq n$ .
2. In the  $k^{th}$  round of allocation, search for the relay and queue index

$$\{r^*, i^*\} \in \arg \max_{1 \leq i \leq n, 1 \leq r \leq R(n)} A_i^{r(k-1)} X_{ij}^r(t),$$

breaking ties in the favor of the smaller relay index, followed by the smaller queue index. If  $A_{i^*}^{r(k-1)} X_{i^*j}^{r^*}(t) > 0$ , goto step 3. Else goto step 4.

3. Allocate channel  $k$  to serve  $R_{r^*i^*}$ , define  $Y_{i^*k}^{r^*}(t) = 1$  and update the value of  $A_{i^*}^{r^*(k)}$  to  $(A_{i^*}^{r^*(k-1)} - X_{i^*j}^{r^*}(t))^+$ . Goto Step 5.

4. Search for the relay and queue index

$$\{r^*, i^*\} \in \arg \max_{1 \leq i \leq n, 1 \leq r \leq R(n)} R_{ri}^{(k-1)} X_{ij}^r(t),$$

breaking ties in the favor of the smaller relay index, followed by the smaller queue index. Allocate channel  $k$  to serve  $R_{r^*i^*}$ , define  $Y_{i^*k}^{r^*}(t) = 1$  and update the value of  $R_{r^*i^*}^{(k)}$  to  $(R_{r^*i^*}^{(k-1)} - X_{i^*j}^{r^*}(t))^+$ .

5. If  $k = n$ , stop, else increment  $k$  by 1 and goto step 2.

We now define a series of events and compute their probabilities.

*Lemma 6.* Under Assumption 2 and the SSG MaxWeight algorithm, let  $E_0$  be the event that the max queue-length of the base-station queues at the end of slot  $t$  is 0. Then,

$$P(E_0^c) = o\left(\frac{1}{n^3}\right).$$

*Proof:* Follows by Lemma 5. ■

*Lemma 7.* Let  $3\epsilon = S_{max} - \lambda$ . Under Assumption 2 and the SSG MaxWeight algorithm, let  $E_1$  be the event that the max arrivals to the base-station queues at the beginning of slot  $t$  is less than  $n(\lambda + \epsilon)$ . Then,

$$P(E_1^c) = o\left(\frac{1}{n^3}\right).$$

*Proof:* Follows by Assumption 2(a).

■

*Lemma 8.* Under Assumption 2(c) and the SSG MaxWeight algorithm, let  $E_2$  be the event that the max arrivals to any relay queue in a given time-slot is less than  $\frac{2nS_{max}}{R(n)}$ . Then,

$$P(E_2^c) = o\left(\frac{1}{n^3}\right).$$

*Proof:* Recall the tie-breaking policy of the SSG MaxWeight rule: initialize the priority order of the relays as  $\{1, 2, \dots, R(n)\}$ . In each round of the allocation process, the relay that is allocated that particular channel is then removed from its current position in the priority order and inserted at the last position to get the new priority order. Consider a particular relay  $r$  which is allocated the  $j^{th}$  channel. It is then pushed to the end of the priority order. In the subsequent rounds of channel allocation, another channel will be allocated to it only if that channel cannot be used at  $S_{max}$  to send packets to any of the other relays which are higher than  $r$  in the priority list. Consider the next  $R(n)/2$  rounds of channel allocation. In each of these rounds, there are at least  $R(n)/2$  relays that have higher priority than relay  $r$ . Then, by Assumption 2(c) for  $\delta = 0.5$ , we have that the probability that relay  $r$  is allocated another channel in the next  $R(n)/2$  rounds of channel allocation is  $o(n^{-4})$ . The result then follows from the union bound over all channels.

■

Let  $E_3 = E_0 \cap E_1 \cap E_2$ . By Lemma 6, 7 and 8,  $P(E_3^c) = o(\frac{1}{n^3})$ . In the following lemma, we prove that the AP-SSG algorithm stabilizes the relay

queues. Then, using the fact that the SSG algorithm is throughput optimal for one hop networks, we prove that the SSG MaxWeight algorithm will also stabilize the relay queues.

*Lemma (3).* Let  $S_{ri} = \sum_{j=1}^n X_{ij}^r Y_{ij}^r$  be the service allocated to queue  $i$  at relay  $r$  by the AP-SSG algorithm. Under Assumption 2(c) and 2(d), let  $E_4$  be the event that

$$\cap_{r,i} \{A_i^r \leq S_{ri}\} \cap \{S_{r^*i^*} \geq A_{i^*}^{r^*} + S_{max}\},$$

where  $\{r^*, i^*\} \in \arg \max_{r,i} R_{ri}(t-1)$ . The event  $E_4$  means that all the arrivals to the relay queues at the beginning of slot  $t$  are served in slot  $t$  and at least one of the longest relay queues is served by at least 1 additional channel. Then,

$$P(E_4^c) = o\left(\frac{1}{n}\right).$$

*Proof:* We condition the proof on  $E_3$ . Pick any  $\delta$  in

$$\left(0, \frac{q_{min}(1 - \lambda - 2\epsilon)}{2S_{max}(2 - q_{min})}\right).$$

Let  $F_m$  be the set of relay queues which received  $m$  packets at the beginning of slot  $t$ . Conditioned on  $E_3$ ,  $|F_m| = 0$  for  $m > \frac{2nS_{max}}{R(n)}$ . Recall that  $3\epsilon = S_{max} - \lambda$ .

Let  $m = \frac{2nS_{max}}{R(n)}$ .

**Case I:**  $|F_m| = |F_m^{(0)}| \geq \delta R(n)$ .

Define  $w_0 = |F_m^{(0)}| - \delta R(n)$ . By Assumption 2(c), we have that after the first  $w_0$  rounds of service,  $|F_m^{(w_0)}| \leq \delta R(n)$  w.p.  $\geq 1 - \delta R(n)o(1/n^3)$ .

Consider the next  $v_0 = \frac{2\delta R(n)}{q_{min}}$  rounds of allocation, By Assumption 2(d), we

have that  $|F_m^{(v_0+w_0)}| = 0$  w.p.  $\geq 1 - o(1/n^3)$ .

**Case II:**  $|F_m| = |F_m^{(0)}| \leq \delta R(n)$ .

Consider the first  $v_0 = \frac{2\delta R(n)}{q_{min}}$  rounds of allocation, By Assumption 2(d), we have that  $|F_m^{(v_0)}| = 0$  w.p.  $\geq 1 - o(1/n^3)$ .

The proof now follows by repeatedly applying the above procedure for  $m = \frac{2nS_{max}}{R(n)} - 1, \frac{2nS_{max}}{R(n)} - 2, \dots, 1$ . As a result, all the new packets are served at the end of

$$\begin{aligned} & n(\lambda + \epsilon) - 2nS_{max}\delta\left(\frac{2}{q_{min}} - 1\right) \\ & < n(1 - \epsilon) \end{aligned}$$

rounds of allocation with probability

$$\geq 1 - P(E_3^c) + \frac{2n^2S_{max}}{R(n)}\left(\delta R(n)o\left(\frac{1}{n^3}\right) + o\left(\frac{1}{n^3}\right)\right).$$

In the remaining  $\epsilon n$  rounds of allocation, by Assumption 2(d), at least one channel serves the longest relay queue with probability  $= o(1/n^3)$ . Therefore,

$$P(E_4^c) = o(1/n).$$

■

*Lemma 9.* Under Assumptions 2 and the Iterative MaxWeight algorithm, for any arrival process with  $\lambda < S_{max}$ , the relay queues are stable for  $n$  large enough.

*Proof:* Let  $R(t+1) = R(t) + A(t) - S(t) + U(t)$  where  $A(t)$ ,  $S(t)$  and  $U(t)$  represent the arrivals, service and unused service respectively. Consider the Lyapunov function  $V(t)$  where  $V(\mathbf{R}(t)) = \|R(t)\|^2$ . We drop the time index for convenience.

$$\begin{aligned}
& E[V(t+1) - V(t) | \mathbf{R}(t)] \\
&= \|R(t+1)\|^2 - \|R(t)\|^2 \\
&= \|R + A - S + U\|^2 - \|R\|^2 \\
&= \|R\|^2 + \|(A - S)\|^2 + 2R(A - S) + \|U\|^2 \\
&\quad + 2\langle U, (R + A - S) \rangle - \|R\|^2 \\
&\leq n^2 S_{max}^2 + 2\langle R, (A - S) \rangle.
\end{aligned}$$

We use the fact that  $U = -(R + A - S)$ , therefore  $\langle U, (R + A - S) \rangle = -\|U\|^2 \leq 0$ .

For the AP-SSG algorithm and the event  $E_4$  defined above,  $P(E_4^c) = o(1/n)$ .

By the definition of event  $E_4$ , we have that

$$E[\langle R, A - S \rangle | \mathbf{R}(t), E_4] \leq -R_{max} S_{max}.$$

Also,

$$E[\langle R, A - S \rangle | \mathbf{R}(t), E_4^c] \leq R_{max} S_{max} n.$$



Therefore,

$$\begin{aligned}
& E[V(t+1) - V(t) | \mathbf{R}(t)] \\
& \leq n^2 S_{max}^2 + 2\langle R, (A - S) \rangle. \\
& \leq n^2 S_{max}^2 - 2R_{max} S_{max} P(E_4) + 2R_{max} S_{max} n P(E_4^c) \\
& \leq n^2 S_{max}^2 - R_{max} S_{max} P(E_4),
\end{aligned}$$

for  $n$  large enough. For  $R_{max} > \frac{n^2 S_{max}^2 - 1/2}{P(E_4) S_{max}}$ , the drift is  $\leq -\frac{1}{2}$ . Therefore, by Foster's theorem, the relay queues are stabilized by the AP-SSG algorithm. Further, by Theorem 5 in [15], the SSG algorithm is throughput optimal for a system consisting of just the relay queues. Since there exists an algorithm (AP-SSG) which can stabilize the relay queues, the SSG algorithm will also stabilize the relay queues.

■

*Theorem (2).* Under Assumption 2, for arrival processes with  $\lambda < S_{max}$ , the SSG MaxWeight algorithm stabilizes the FD-w/oDL system, i.e., the markov chain  $\{\mathbf{Q}(t), \mathbf{R}(t), \mathbf{A}(t)\}$  is positive recurrent for  $n > n_0$  where  $n_0$  is a function of  $\lambda$ .

*Proof:* The proof follows from Lemma 5 and Lemma 9.

■

### A.1.2 HD-wDL

This proof proceeds in the following three steps. Please refer to [72] for the complete proof.

1. We first prove that under Assumption 3, there are no arrivals to the relays at the beginning of a slot with probability  $= o(1/n^2)$ .
2. We then prove that with high probability, the maximum queue-length in the system does not increase in any time-slot.
3. Next, we prove that there exists a constant  $k_0$  such that in  $k_0$  consecutive time-slots, the maximum queue-length in the system decreases by 1 with probability  $\geq 1/2$ . We use the proof technique used in Lemma 8 in [15] to get this result.
4. Finally, We prove the stability of the system by constructing a Markov Chain of the maximum queue-length of the system. We then use Theorems 2 and 3 from [16] and Lemma 4 to prove stability of this Markov Chain, thus proving the stability of the HD-wDL system.

## A.2 Performance Analysis

In this section, we analyze the rate function for the small buffer overflow probability of the BackPressure algorithm, the SSG MaxWeight algorithm, the ILQF MaxWeight algorithm and the ILQF BackPressure algorithm for the FD-W/oDL model.

### A.2.1 BackPressure

We first show that the BackPressure algorithm has a zero rate for the small buffer overflow event. The proof follows on the same lines as the proof of Theorem 3 in [15]. In [15], it was proved that the maximum queue-length increases with at least a constant probability in each slot. We prove the same result for the backpressure value of the base-station queues and use the backpressure values as a lower bound for the queue-lengths to prove the desired result. Please refer to [72] for the complete proof.

### A.2.2 SSG MaxWeight

The proofs for performance of the ILQF BackPressure algorithm, the ILQF MaxWeight algorithm and the SSG MaxWeight algorithm for the FD-w/oDL system proposed in Section 2.4 work in a sequential manner. We divide the set of queues into two sets: the base-station queues and the relay queues. Even though the two sets of queues are coupled, surprisingly, they can be analyzed in a sequential manner to provide performance guarantees on all the queue-lengths in the system.

For the ILQF BackPressure algorithm, we analyze the relay queues first and prove that they are all empty with probability  $\approx e^{-nc}$  for some  $c > 0$ . We observe that at the base-station, the ILQF backpressure algorithm tries to serve queues with the highest backpressure values which are not always queues with maximum queue lengths. However, if the relay queues are all empty, the two sets are the same. We use this fact to analyze the maximum base-station

queue length.

For the ILQF MaxWeight algorithm and the SSG MaxWeight algorithm, we analyze the base-station queues first and use that result to analyze the relay queues.

The analysis for each set of queues is carried out in the following steps:

1. We first show that for the set of queues that we are analyzing (either the relay queues or the base-station queues), the maximum queue length increases in a slot with a very small probability ( $\leq e^{-nc}$ ).
2. Using Step 1 and Lemma 8 in [15], we show that there exists a constant  $k_0$  such that in  $k_0$  consecutive time-slots, with probability at least  $1/2$ , the maximum queue length decreases by 1.
3. To compute the stationary distribution of the maximum value of queues in this set, we construct a Markov Chain  $Y^{(n)}(t)$  which has the following properties:

$$\begin{aligned} P(Y^{(n)}(t+1) = (Y^{(n)}(t) - 1)^+) &= 1/2 \\ P(Y^{(n)}(t+1) = Y^{(n)}(t) + \chi(n)) &= e^{-nc} \\ P(Y^{(n)}(t+1) = Y^{(n)}(t)) &= 1/2 - e^{-nc}. \end{aligned}$$

For the relay queues,  $\chi(n) = k_0 n$ . We prove that for  $f(n) = e^{-nc}$  for some  $c > 0$ , we have that,

$$\liminf_{n \rightarrow \infty} \frac{-1}{n} \log P\left(Y^{(n)}(0) > b\right) \geq (b+1)c.$$

For the base-station queues,  $\chi(n) = k_0$ . Using Theorem 4 in [16], we have that,

$$\liminf_{n \rightarrow \infty} \frac{-1}{n} \log P\left(Y^{(n)}(0) > b\right) \geq (b+1)c.$$

4. We use Theorem 3 in [16] to prove that the maximum queue length in the set of interest is stochastically dominated by the process  $Y^{(n)}(t)$  for the corresponding value of  $\chi(n)$ . We then use the stationary distribution of  $Y^{(n)}(t)$  to get the desired result.

For the SSG MaxWeight algorithm, we first focus on the base-station queues and find the probability that in the steady state, the maximum queue-length is greater than  $b$  at the beginning of a slot. Conditioned on the fact that the longest base-station queue has  $b$  packets, at the end of time-slot  $t-1$ , not more than  $b+1$  packet can arrive to any particular relay queue at the beginning of slot  $t+1$ . Using this, we find the probability that in the steady state, all relay queues have less than  $b$  packets at the end of a time-slot for all integers  $b \geq 0$ .

### *Base-station Queues*

*Lemma 10.* Fix a value of  $\epsilon \in (0, 1-p)$ . Define

$$\xi_B(t) =: \max_{1 \leq i \leq n} Q_i(t).$$

Then,

$$P(\xi_B(t) > \xi_B(t-1)) \leq e^{-c_B n^2 + k(\epsilon)n} + e^{-nH(p|p+\epsilon)}.$$

*Proof:* Consider the event  $E$  that

$$\sum_{i=1}^n A_i(t) \leq (p + \epsilon)n.$$

Then,

$$P(E^c) \leq e^{-nH(p|\epsilon)}.$$

We condition the rest of the proof on the event  $E$ .

Let  $F$  denote the set of queues whose length is  $\xi_B(t-1) + 1$  after incorporating arrivals for that slot. Let  $F^{(i)}$  denote the updated set after  $i$  rounds of channel allocation. If  $\xi_B(t) > \xi_B(t-1)$ , then there exist at least  $n(1 - p - \epsilon)$  channels that were not used.

$$\begin{aligned} P(n(1 - p - \epsilon) \text{ unused channels}) &= (1 - q_2)^{\tilde{R}n^2(1-p-\epsilon)} \\ &= e^{-c_B n^2}, \end{aligned}$$

where  $c_B = \tilde{R}(1 - p - \epsilon) \log \frac{1}{1-q_2}$ . Therefore,

$$P(\xi_B(t) > \xi_B(t-1)) \leq e^{-c_B n^2 + k(\epsilon)n} + e^{-nH(p|\epsilon)}.$$

■

We now prove there exists a constant  $k_0$  such that the maximum relay queue-length decreases by 1 in  $k_0$  consecutive time-slots with probability  $\geq 1/2$ .

*Lemma 11.* We can find  $k_0$  such that

$$P(\xi_R(t + k_0) = \xi_R(t) - 1) \geq \frac{1}{2}.$$

*Proof:* The proof follows from Lemma 8 in [15] and Lemma 10 as stated above. ■

The following theorem uses the same proof technique as Theorem 5 in [16] to compute a bound on the rate function for the small buffer overflow event for the base-station queues using Lemma 10 and 11.

*Theorem (5a).* Under Assumption 4, for the SSG MaxWeight algorithm, for any  $\epsilon \in (0, 1 - p)$ ,

$$\liminf_{n \rightarrow \infty} \frac{-1}{n} \log P \left( \max_{1 \leq i \leq n} Q_i(0) > b \right) \geq c(b + 1).$$

Where,

$$c = H(p|p + \epsilon) > 0.$$

*Proof:* Using Lemma 10 and Lemma 11 as stated above and by Theorem 5 in [16]. ■

### *Relay Queues*

In the following theorem we use the same proof technique as was used to compute the rate function of the SSG algorithm in [16] with the additional step that we use the fact that the base-station queues have less than  $b$  at the

end of every time-slot with an exponentially large probability. Conditioned on this event, the maximum number of packets that arrive to any relay queue in a time-slot is  $b + 1$ . This is an important step in this proof because potentially  $nS_{max}$  packets can arrive to a particular relay queue in a given time-slot and it is not possible to serve all of them in that time-slot and therefore the maximum queue-length in the relay queues can increase in a time-slot.

*Theorem (5b).* Under Assumption 4, for the SSG MaxWeight algorithm, for any  $\epsilon \in (0, 1 - p)$  and

$$\delta \in \left(0, \frac{q_3(1 - p - \epsilon)}{2 - q_3}\right),$$

$$\liminf_{n \rightarrow \infty} \frac{-1}{n} \log P \left( \max_{1 \leq i \leq n, 1 \leq j \leq k} R_{ik}(0) > b \right) = (b + 1)c_R,$$

where,

$$c_R \geq \min \left( H(p|p + \epsilon), \delta \log \frac{1}{1 - q_3}, \frac{2\delta H(q_3|\frac{q_3}{2})}{q_3} \right).$$

*Proof:* Consider the event  $E_5$  that  $\xi_B(t - 1) = b$ . This implies that all the base-station queues had less than  $b$  packets at the end of time-slot  $t - 1$ . Then from Theorem 5a, we have that,

$$P(E_5^c) \leq (b + 1)s(n)e^{-nH(p|p + \epsilon)},$$

where  $s(n)$  is a sub-exponential function of  $n$ . We condition the rest of the proof on the event  $E_5$ .

Conditioned on  $E_5$ , the maximum possible arrivals to any relay queue at the beginning of slot  $t$  is  $b + 1$ . Therefore, using the same steps as in Theorem 5



in [16], we have that, for any  $\epsilon \in (0, 1 - p)$  and

$$\delta \in \left(0, \frac{q_3(1-p)}{2-q_3}\right),$$

$$\liminf_{n \rightarrow \infty} \frac{-1}{n} \log P \left( \max_{1 \leq i \leq n, 1 \leq j \leq k} R_{ik}(0) > b \right) = (b+1)c_R,$$

where,

$$c_R \geq \min \left( H(p|p+\epsilon), \delta \log \frac{1}{1-q_3}, \frac{2\delta H(q_3|\frac{q_3}{2})}{q_3} \right).$$

■

### A.2.3 ILQF BackPressure

For the ILQF BackPressure algorithm, we first focus on the relay queues and find the probability that in the steady state, they are all empty at the beginning of a slot. We observe that at the base-station, the iterative back-pressure algorithm tries to serve queues with the highest backpressure values which are not always queues with maximum queue-lengths. However, conditioned on the fact that the relay queues are all empty, the two sets are the same. This allows us to bound the probability that the maximum base-station queue-length at the end of a time-slot is  $> b$ . Please refer to [72] for the complete proof.

### A.2.4 ILQF MaxWeight

Similar to the analysis of the SSG MaxWeight algorithm, we first focus on the base-station queues and find the probability that in the steady state,

they have less than  $b$  packets at the beginning of a slot. Conditioned on the fact that the base-station queues have less than  $b$  packets at the end of time-slot  $t - 1$ , not more than  $b + 1$  packet can arrive to any particular relay queue at the beginning of slot  $t + 1$ . Using this, we find the probability that in the steady state, all relay queues are empty at the end of a time-slot.

### A.3 $k$ -hop Stability

We consider a  $k$ -hop full-duplex feed-forward network with 1 base-station,  $k - 1$  layers of relays and  $n$  users. The relays in the first layer of relays receive packets from the base-station and the relays in the  $k^{th}$  layer forward received packets to the users. A relay in the  $l^{th}$  layer (for  $2 \leq l \leq k - 1$ ) receives packets from the  $(l - 1)^{th}$  layer of relays and forwards them to the next layer. See Figure A.1 for an example of such a network.

We use the following notation for this proof.

- $A_i(t)$  = the number of arrivals for user  $i$  at the base-station at the beginning of time-slot  $t$ .
- $Q_i(t)$  = The queue length of user  $i$  at the BS (measured at the end of the time-slot).
- $R_{(l),ri}(t)$  = The queue length of user  $i$  at relay  $r$  at layer  $l$  (measured at the end of the time-slot).

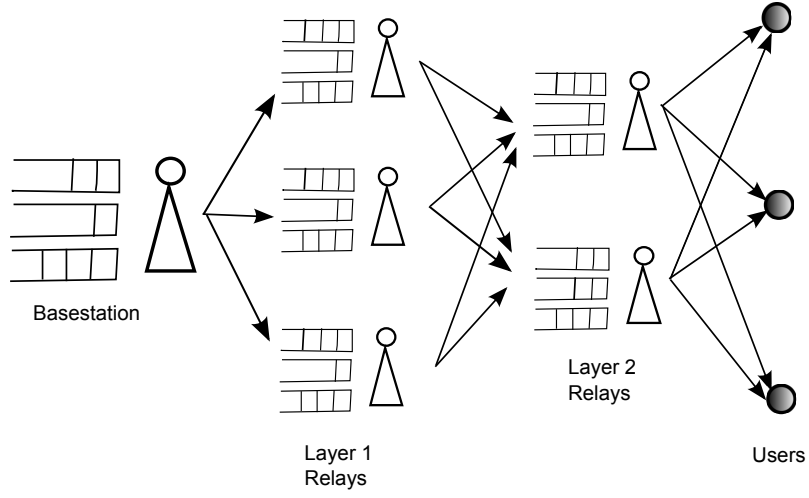


Figure A.1: An illustrative example of a 3-hop feed-forward relay network with 2 layers of relays and 3 users.

- $\mathbf{R}_{(l)}(t) = \{R_{(l)ri}(t) : \forall r; 1 \leq i \leq n\}$  : The vector of queue lengths at the relays at layer  $l$ .

The  $k$ -hop version of the SSG MaxWeight algorithm is as follows:

In each time-slot, for each hop, sequentially allocate channels to queues in the following manner: first allocate channel  $S_1$  to the maximum weight queue, i.e., the queue with largest queue-length channel-rate product. Then update the queue length based on the number of packets that are drained due to this allocation, and proceeds sequentially to the next channel (and so on).

For simplicity, we provide a proof of the stability of SSG MaxWeight under the following assumptions.

**Assumption 5:** ( $k$ -hop Stability)

- The base-station can forward packets to all relays in the first layer of relays. Each relay in layer  $l$  for every  $l \in \{1, \dots, k-2\}$  can forward packets to all relays in the next layer (layer  $l+1$ ). Each relay in layer  $k-1$  can communicate with all the users.
- Bernoulli Arrivals and ON-OFF Channels
  - $A_i(t) = \text{Bernoulli}(p)$  i.i.d. across users and time-slots.
  - All channels are  $\text{Bernoulli}(q)$  i.i.d. across channels, time-slots, relays and users.
- Linearly Scaling Relays: We assume that the  $l^{\text{th}}$  layer of relays has  $v_l n$  relays for some constant  $v_l > 0$ .

**Theorem 27.** *Under Assumption 5, the  $k$ -hop system is stabilized by the SSG MaxWeight algorithm.*

*Proof:* The stability of the base-station queues follows from Lemma 5. In addition, by applying Theorem 4 for  $\chi(n) = 1$  and  $f(n) = e^{-nc_1}$  for some  $c_1 > 0$ , we have that

$$P(\max_i Q_i(t) > 0) \leq 4e^{-nc_1},$$

for all  $t$ .

Let  $F_1$  be the event that  $\max_i Q_i(t-1) = 0$ . Therefore, we have that,  $P(F_1^c) \leq 4e^{-nc_1}$ . The rest of this proof is conditioned on  $F_1$ . Consider the queues at the relays of the first layer. In each round of channel allocation under

the SSG MaxWeight algorithm, the probability that the channel cannot serve the currently longest queue (updated to reflect previous rounds of allocations) is  $(1 - q)^{v_2 n}$ . Therefore with probability  $> n(1 - q)^{v_2 n}$ , in a given time-slot, each channel serves the currently longest queue (updated to reflect previous rounds of allocations). Since the total arrivals to the relay queues at the first hop in a time-slot is less than  $\leq n$ , with probability  $\geq 1 - 4e^{-nc_1} - n(1 - q)^{v_2 n}$ , the maximum queue-length at the first layer of relays does not increase in a time-slot. Therefore, we have that,

$$P(\max_{r,i} R_{(1)ri}(t + 1) = \max_{r,i} R_{(1)ri}(t) + 1) \leq 4e^{-nc_1}.$$

Using this and Lemma 8 in [16], we can find  $k_0$  such that,

$$P(\max_{r,i} R_{(1)ri}(t + 1) = \max_{r,i} R_{(1)ri}(t) - 1) \geq \frac{1}{2}.$$

The stability of the relay queues at the first layer then follows using the Lyapunov function  $Lyap(\mathbf{R}_{(1)}(t)) = \max_{r,i} R_{(1)ri}(t)$ .

In addition, using Theorem 4, we have that,

$$P(\max_{r,i} R_{(1)ri}(t) > 0) \leq 16k_0 e^{-nc_1} + 4nk_0(1 - q)^{v_2 n}.$$

For the queues at the  $l^{th}$  layer for  $2 \leq l \leq k - 2$ , the proof of stability follows on the same lines as the proof of stability for relay queues at layer 1. For layer  $l$ , the proof follows by conditioning on the event that the queues at the base-station and relay layers 1 to  $l - 1$  are empty in the previous  $l$  time-slots.

The stability of the relay queues at layer  $l$  follows from Lemma 9, thus completing the proof of Theorem 27.

■

# Appendix B

## Proofs from Chapter 3

### B.1 Stability

*Proof.* (of Theorem 8) If  $\lambda > 1$ , then the mean number packet arrivals to the system in a given time-slot is more than the maximum number of packets that can be served by the base-station in a given time-slot ( $= nC_{max}$ ). Hence the system is unstable under any scheduling algorithm.  $\square$

We now prove that if the load on the system  $\lambda < 1$ , the DIST algorithm stabilizes the system.

*Lemma 12.* The arrival process at the ANs is bounded by  $(d + 1)\kappa(n)$ , i.e.,

$$\max_{1 \leq i \leq n, 1 \leq m \leq M(n)} A_i^m(t) \leq (d + 1)\kappa(n) \text{ for all } t.$$

*Proof.* In time-slot  $t$ , the BS only sends those packets which arrived at the BS at the beginning of time-slot  $t$  to the ANs. In addition, at the end of time-slot  $t - 1$ , the ANs delete all packets in their queues which arrived at the BS before time-slot  $t - d - 1$ . Since the arrival process at the base-station is bounded by  $\kappa(n)$  by Assumption (a), we have that the arrival process at the ANs is also bounded by  $(d + 1)\kappa(n)$ .  $\square$

*Lemma 13.* Let  $U(t)$  be the set of users whose location is known at the beginning of time-slots  $t$  and  $t + 1$ . Consider the set of all packets for users  $u \in U(t)$  which are received and queued at least one AN at the beginning of time-slot  $t$ . A packet for user  $u$  in this set is said to be “lost” in time-slot  $t$  if at the beginning of time-slot  $t + 1$ , it is neither received by the user  $u$  nor by at least one AN connected to user  $u$  in time-slot  $t + 1$ . Let  $E_1$  be the event that in time-slot  $t$ , at most  $\sigma n$  of the packets are “lost”. For the DIST-AN algorithm,

$$P(E_1^c) \leq o\left(\frac{1}{n^2}\right).$$

*Proof.* Let  $q_2 := q_2^{(C_{max})}$ . Consider the event  $E_2$  that for any (AN, user) pair, in a time-slot, the number of channels that have channel rate  $C_{(max)}$  is at least  $\frac{nq_2}{2}$ . By Assumption (a),

$$P(E_2^c) \leq o\left(\frac{1}{n^2}\right).$$

The rest of this proof is conditioned on  $E_2$  for all (AN, user) pairs. Let  $q_3 := q_3^{(C_{max})}$ . Consider the event  $E_3$  that from an AN to another AN within its communication radius, in a time-slot, the number of channels that have channel rate  $C_{(max)}$  is at least  $\frac{nq_3}{2}$ . By Assumption (a),

$$P(E_3^c) \leq o\left(\frac{1}{n^2}\right).$$

The rest of this proof is conditioned on  $E_3$  for all such (AN, AN) pairs. Let  $E_4$  be the event that not more than  $n^\nu$  users are connected to any 1 AN where  $\nu < 1$  is discussed in Section 3.2. Then, we have that,

$$P(E_4^c) \leq e^{-bn},$$



for some  $b > 0$ . The rest of this proof is conditioned on  $E_4$ . By Lemma 12, the arrival process at the ANs is bounded by  $(d+1)\kappa(n)$ . Therefore, every AN has at most  $(d+1)n^\nu\kappa(n)$  packets to send in a time-slot and can use at most  $k_1(d+1)n^\nu\kappa(n)\log n$  channels to do so since by our assumption in Section 3.2, each AN can communicate with at most  $k_1\log n$  other ANs. Therefore, each packet is sent to either  $k_1\log n$  other ANs or one user. So, for each AN, in any round of channel allocation, there are at least  $\frac{nq_2}{2} - k_1(d+1)n^\nu\kappa(n)\log n$  channels that have channel rate  $C_{max}$  for every user and  $\frac{nq_3}{2} - k_1(d+1)n^\nu\kappa(n)\log n$  channels that have channel rate  $C_{max}$  for every AN within its communication radius. Therefore,

$$P(\text{AN } m \text{ uses channel } j | E_2, E_3, E_4) \leq \frac{k_1(d+1)n^\nu\kappa(n)\log n}{\frac{nq^{(c)}}{2} - k_1(d+1)n^\nu\kappa(n)\log n},$$

where  $q = \min\{q_2, q_3\}$ . Let  $p$  be a packet that AN  $m$  transmits on channel  $j$  in time-slot  $t$ . Let  $E_5$  be the event that this packet reaches its destination without interference from other ANs in  $I_m$ . Then, irrespective of all other channel allocations by other ANs, we have that,

$$P(E_5^c | E_2, E_3, E_4) \leq \frac{k_1(d+1)n^\nu\kappa(n)\log n}{\frac{\eta nq^{(c)}}{2} - k_1(d+1)n^\nu\kappa(n)\log n}.$$

At the beginning of time-slot  $t$ , there are at most  $(d+1)C_{max}n$  packets for users  $u \in U(t)$  that are received and queued at the ANs. Therefore, for  $n$  large

enough,

$$\begin{aligned}
P(E_1^c | E_2, E_3, E_4) &\leq \binom{n(d+1)C_{max}}{n\sigma} \\
&\quad \times \left( \frac{k_1(d+1)n^\nu \kappa(n) \log n}{\frac{nq^{(c)}}{2} - k_1(d+1)\kappa(n)n^\nu \log n} \right)^{\sigma n} \\
&= o\left(\frac{1}{n^2}\right),
\end{aligned}$$

by Assumption (a) and the assumptions made in Section 3.2. Therefore, the result follows.  $\square$

*Lemma 14.* All arrivals to the base-station at the beginning to time-slot  $t$  for users whose location information is available are sent to the ANs by the in  $(\lambda + \sigma)n$  rounds of channel allocation with probability  $\geq 1 - o(1/n)$ .

*Proof.* Let  $E_6$  be the event that not more than  $n(\lambda + \sigma)C_{max}$  packets arrive at the base-station at the beginning of time-slot  $t$ . By Assumption (a), we have that,

$$P(E_6^c) \leq o\left(\frac{1}{n}\right).$$

The rest of this proof is conditioned on the event  $E_6$ . Let  $E_7$  be the event that in the first  $(\lambda + \sigma)n$  rounds of channel allocation,  $(\lambda + \sigma)n$  channels can be used by the base-station to send packets from the longest BS queue (after updating after previous channel allocations) to the ANs at rate  $C_{max}$ . Under Assumption (a), the probability that a particular channel  $k$  can be used to forward packets from the longest queue to the ANs connected to the

corresponding user is  $\geq o(1/n^2)$ . Taking a union bound over all channels, we have that,

$$P(E_7^c) \leq o\left(\frac{1}{n}\right).$$

Conditioned on  $E_7$ , all packets which arrived at the base-station at the beginning of time-slot  $t$  will be served in the first  $(\lambda + \sigma)n$  rounds of channel allocation. Therefore, all arrivals to the base-station at the beginning to time-slot  $t$  are sent to the ANs by the in  $(\lambda + \sigma)n$  rounds of channel allocation with probability  $\geq 1 - o\left(\frac{1}{n}\right)$ .  $\square$

*Lemma 15.* For any  $t$ ,

1.  $\max_{1 \leq u \leq n} F_u(t) \leq \kappa(n)$
2. In a time-slot  $t$ .

$$P\left(|F(t)|_1 \leq 4n\sigma \geq 1 - o\left(\frac{1}{n}\right)\right),$$

where  $\sigma$  is as defined in Lemma 13.

*Proof.* The first part of the lemma follows from the fact that the arrival process for a user  $u$  at the BS is bounded by  $\kappa(n)$  and therefore, the number of packets for a user  $u$  that arrived before time-slot  $t - d$ , but could not be received by the corresponding users by time-slot  $t$  is  $\leq \kappa(n)$ .

This proof is conditioned on the event  $E_7$  defined in Lemma 14. We compute

the probability of the event  $E_8$  that in a time-slot, more than  $\frac{n\sigma}{\kappa(n)}$  users are not connected to the ANs or their locality information is not available. The probability that a user's information is not known is  $\epsilon(n)$ , where  $\epsilon(n)\kappa(n) = o(1)$ . By the Chernoff bound, we have that,

$$P(E_8^c) = o\left(\frac{1}{n}\right).$$

The rest of this proof is conditioned of  $E_8$  for time-slots  $t-d$  and  $t-d+1$ . The number of packets that arrived at the BS at beginning of time-slot  $t-d-1$  for users that were not connected to the ANs in either time-slot  $t-d$  or  $t-d+1$  or both is  $2n\sigma$ .

Now consider the packets that arrived at the BS at beginning of time-slot  $t-d-1$  for users that were connected to the ANs in both time-slots  $t-d$  and  $t-d+1$ . Conditioned on the event  $E_1$  defined in Lemma 17, the number of packets “lost” in time-slots  $t-d$  is less than  $n\sigma$ . If a packets is not lost in time-slot  $t-d$ , it either reaches the intended user by the end of time-slot  $t-d$  or an AN connected to the user in time-slot  $t-d+1$ .

Consider the set of packets for users which are received by at least 1 AN connected to the corresponding user in time-slot  $t-d+1$ . Since at most  $\sigma n$  packets are lost in time-slot  $t-d+1$ , all but at most  $\sigma n$  of such packets reach the corresponding users by the end of time-slot  $t-d+1$ .

From this, we conclude that, for users that are connected to ANs in time-slot  $t-d$  and  $t-d+1$ , all but  $2\sigma n$  of the arrivals at the BS in time-slot  $t-d-1$  are received by the users by the end of time-slot  $t-d+1$ . Therefore,

conditioned on  $E_1$  and  $E_8$  for time-slots  $t - d$  and  $t - d + 1$ , we have that,

$$P\left(|F(t)|_1 \leq 4n\sigma\right) \leq o\left(\frac{1}{n}\right).$$

□

*Lemma 16.* Let  $S_i(t) = \sum_{j=1}^n X_{i,j}(t)Y_{i,j}(t)$  be the service allocated to queue  $i$  at the base-station by the DIST algorithm in time-slot  $t$ . Dropping the time index for simplicity, let  $G_9$  be the event that

$$\cap_i \{F_i \leq S_i\} \cap \{S_{i^*} \geq F_{i^*} + 1\},$$

where  $i^* \in \arg \max_i Q_i(t - 1)$ . The event  $G_9$  means that all the new arrivals and the feedback to the base-station queues at the beginning of slot  $t$  are served in slot  $t$  and at least one of the longest queues is served by at least 1 additional channel. Then,

$$P(G_9^c) = o\left(\frac{1}{n}\right).$$

*Proof.* The proof is conditioned on the event  $G_7$ . Let  $q_{min}^{(d)} = \min_{i,j} X_{i,j}(t) = 1$ . Pick any  $\delta$  in

$$\left(0, \frac{q_{min}^{(d)}(\sigma)}{2(2 - q_{min}^{(d)})}\right).$$

Let  $F_r$  be the set of queues which received  $r$  new packets at the beginning of slot  $t$ . We know that  $|F_r| = 0$  for  $r > \kappa(n)$ . Let  $r = \kappa(n)$ .

**Case I:**  $|F_r| = |F_r^{(0)}| \geq \delta n$ .

Define  $w_0 = |F_r^{(0)}| - \delta n$ . By Assumption (a), we have that after the first  $w_0$

rounds of service,  $|F_r^{(w_0)}| \leq \delta n$  w.p.  $\geq 1 - \delta n o(1/n^3)$ .

Consider the next  $v_0 = \frac{2\delta n}{q_{min}^{(d)}}$  rounds of allocation, By Assumption (a), we have that  $|F_r^{(v_0+w_0)}| = 0$  w.p.  $\geq 1 - o(1/n^3)$ .

**Case II:**  $|F_r| = |F_r^{(0)}| \leq \delta n$ .

Consider the first  $v_0 = \frac{2\delta n}{q_{min}^{(d)}}$  rounds of allocation, By Assumption (a), we have that  $|F_m^{(v_0)}| = 0$  w.p.  $\geq 1 - o(1/n^3)$ .

Conditioned on  $G_7$ , there are  $6\sigma n$  channels unused by the BS-AN links. The proof now follows by repeatedly applying the above procedure for  $r = \kappa(n), \kappa(n) - 1, \dots, 1$ . As a result, all new feedback arrival packets are served at the end of the next  $5\sigma$  rounds of allocation with probability

$$\geq 1 - \frac{2n^2 S_{max}}{n} \left( \delta n o\left(\frac{1}{n^3}\right) + o\left(\frac{1}{n^3}\right) \right).$$

In the remaining  $\sigma n$  rounds of allocation, by Assumption (a), at least one channel serves the longest relay queue with probability  $= o(1/n^3)$ . Therefore,

$$P(G_9^c) = o\left(\frac{1}{n}\right).$$

□

*Proof.* (of Theorem 9) Consider the Lyapunov function  $V(t)$  where  $V(\mathbf{Q}(t), \mathbf{A}(t)) =$

$\|Q(t)\|^2$ . We drop the time index for convenience.

$$\begin{aligned}
& E[V(t+1) - V(t) | \mathbf{Q}(t)] \\
&= \|Q(t+1)\|^2 - \|Q(t)\|^2 \\
&= \|Q + F - S + U\|^2 - \|Q\|^2 \\
&= \|Q\|^2 + \|(F - S)\|^2 + 2Q(F - S) + \|U\|^2 \\
&\quad + 2\langle U, (Q + F - S) \rangle - \|Q\|^2 \\
&\leq n^2 C_{max}^2 + 2\langle Q, (F - S) \rangle.
\end{aligned}$$

We use the fact that  $U = -(Q + F - S)$ , therefore  $\langle U, (Q + F - S) \rangle = -\|U\|^2 \leq 0$ .

For the DIST algorithm and the event  $G_9$  defined above,  $P(G_9^c) = o(1/n)$ . By the definition of event  $G_9$ , we have that

$$E[\langle Q, F - S \rangle | \mathbf{Q}(t), G_9] \leq -Q_{max}.$$

Also,

$$E[\langle Q, F - S \rangle | \mathbf{Q}(t), G_9^c] \leq Q_{max} C_{max} n.$$

Therefore,

$$\begin{aligned}
& E[V(t+1) - V(t) | \mathbf{Q}(t)] \\
&\leq n^2 C_{max}^2 + 2\langle Q, (F - S) \rangle. \\
&\leq n^2 C_{max}^2 - 2Q_{max} P(G_9) + 2Q_{max} C_{max} n P(G_9^c) \\
&\leq n^2 C_{max}^2 - Q_{max} P(G_9),
\end{aligned}$$

for  $n$  large enough. For  $Q_{max} > \frac{n^2 C_{max}^2 - 1/2}{P(G_9)}$ , the drift is  $\leq -\frac{1}{2}$ . Therefore, by Foster's theorem, the queues are stabilized by the DIST algorithm.  $\square$

## B.2 Performance

*Proof.* (of Theorem 10) Let a packet for a mobile user  $u$  be sent to AN  $m$  in time-slot  $t$ . Let  $E$  be the event that the user is not connected to AN  $m$  in the next  $b$  time-slots.

$$P(E) \geq (\min\{\mu_1, \mu_2\})^b.$$

Conditioned on  $E$ , the packets cannot reach the user before time-slot  $t + b$ . Hence the result follows.  $\square$

*Lemma 17.* Recall that  $U(t)$  is the set of users whose location is known at the beginning of time-slot  $t$ . Consider the set of all packets for users  $u \in U(t)$  which are received and queued at least one AN at the beginning of time-slot  $t$ . As defined in Lemma 13, a packet for user  $u$  in this set is said to be “lost” in time-slot  $t$  if by the end of time-slot  $t$ , it is neither received by the user  $u$  nor at least one AN connected to user  $u$  in time-slot  $t + 1$ . Let  $\gamma > 0$  be a constant. Let  $G_1$  be the event that in a time-slot, at most  $\gamma n$  are “lost”. Under Assumption (b), for the DIST-AN algorithm,

$$\begin{aligned} P(G_1^c) &\leq o(e^{-n}) + \exp\left(-nH\left(\frac{q_2^{(C)}}{2} | q_2^{(C)}\right)\right) \\ &\quad + \exp\left(-nH\left(\frac{q_3^{(C)}}{2} | q_3^{(C)}\right)\right) + e^{-bn}. \end{aligned}$$



*Proof.* Consider the event  $G_2$  that for any (AN, user) pair, in a time-slot, the number of channels that have channel rate  $C$  is at least  $\frac{nq_2^{(C)}}{2}$ . Since channels are i.i.d. across users and ANs,

$$P(G_2^c) \leq \exp \left( -nH \left( \frac{q_2^{(C)}}{2} | q_2^{(C)} \right) \right).$$

The rest of this proof is conditioned on  $G_2$  for all (AN, user) pairs. Consider the event  $G_3$  that for any AN in the communication radius of an AN, in a time-slot, the number of channels that have channel rate  $C$  is at least  $\frac{nq_3^{(C)}}{2}$ . Since channels are i.i.d. across users and ANs,

$$P(G_3^c) \leq \exp \left( -nH \left( \frac{q_3^{(C)}}{2} | q_3^{(C)} \right) \right).$$

The rest of this proof is conditioned on  $G_3$  for all such (AN, AN) pairs. Let  $G_4$  be the event that not more than  $n^\nu$  users are connected to any 1 AN where  $\nu$  is as defined in Section 3.2. By the assumptions in Section 3.2,

$$P(G_4^c) \leq e^{-bn},$$

for some  $b > 0$ . The rest of this proof is conditioned on  $G_4$ .

The arrival process at the ANs is bounded by  $K$ . Therefore, every AN has at most  $Kn^\nu$  packets to send in a time-slot and can use at most  $Kn^\nu$  channels to do so. So, for each AN, in any round of channel allocation, there are at least  $\frac{nq_2^{(C)}}{2} - Kn^\nu$  channels that have channel rate  $C$  for every user. Therefore,

$$P(\text{AN } m \text{ uses channel } j | G_2, G_3, G_4) \leq \frac{Kn^\nu}{\frac{nq_2^{(C)}}{2} - Kn^\nu}.$$

where  $q^{(c)} = \min\{q_2^{(c)}, q_3^{(c)}\}$ . Let  $p$  be a packet that AN  $m$  transmits on channel  $j$  in time-slot  $t$ . Let  $G_5$  be the event that this packet reaches its destination without interference from other ANs in  $I_m$ . Then, irrespective of all other channel allocations by other ANs, we have that,

$$P(G_5^c | G_2, G_3, G_4) \leq \frac{n^\nu K n^\beta}{\frac{\eta n q_2^{(C)}}{2} - K n^\nu}.$$

Let  $G_6$  be the event that  $G_5^c$  occurs for at most  $\gamma n$  packets. Then we have that for  $n$  large enough,

$$\begin{aligned} P(G_5^c | G_2, G_3, G_4) &\leq \binom{n(d+1)C_{max}}{n\gamma} \left( \frac{n^\beta K n^\nu}{\frac{\eta n q_2^{(C)}}{2} - K n^\nu} \right)^{\gamma n} \\ &\leq 2^{nH(\gamma/(d+1)C_{max})} \left( \frac{K n^\kappa}{\frac{n q^{(C)}}{2} - K n^\nu} \right)^{\gamma n} \\ &\leq 2^{nH(\gamma/(d+1)C_{max})} C_1 e^{-(1-\beta-\nu)\gamma n \log n} \\ &= o(e^{-n}). \end{aligned}$$

Therefore,

$$\begin{aligned} P(G_1^c) &\leq o(e^{-n}) + \exp \left( -nH \left( \frac{q_2^{(C)}}{2} | q_2^{(C)} \right) \right) \\ &\quad + \exp \left( -nH \left( \frac{q_3^{(C)}}{2} | q_3^{(C)} \right) \right) + e^{-bn}. \end{aligned}$$

□

*Lemma 18.* Fix

$$\gamma \in \frac{1}{7} \left( 0, \min \left( \frac{p_0}{K}, \frac{1 - \sum_{k=1}^K p_k \lceil \frac{k}{C} \rceil}{\lceil \frac{K}{C} \rceil C} \right) \right).$$

and let  $M$  be the set difference between the probability simplex in  $K$  dimensions and an  $\gamma$  ball around the probability vector  $p$ . Let

$$\tau := \inf_M \sum_{k=0}^K z_k \log \frac{z_k}{p_k}.$$

Let  $G_6$  be the event that all arrivals to the base-station at the beginning to time-slot  $t$  are sent to the ANs by the in  $(\lambda + 2\gamma)n$  rounds of channel allocation. Then, for a positive constant  $\rho < 1$ ,

$$P(G_6^c) \leq e^{-n\tau(1-\rho)} + \exp(-n\gamma^2/2).$$

*Proof.* By Sanov's theorem, we have that the load on the system in time-slot  $t \leq (\lambda + \gamma)$  with probability  $\geq e^{-n\tau(1-\rho)}$  for any  $\rho < 1$ . We condition the rest of the proof of this event. By Assumption (b), the probability that channel  $k$  cannot be used at rate  $C$  to serve the longest queue at the base-station updates after  $k - 1$  rounds of allocation is  $\geq (1 - q_1^{(C)})^{2\log n} = o(1)$ . Define  $\epsilon'(n) = (1 - q_1^{(C)})^{2\log n}$

$$P(G_6^c) = \exp(-nD(\gamma||\epsilon'(n))),$$

where

$$D(\gamma||\epsilon(n)) = \gamma \log \frac{\gamma}{\epsilon'(n)} + (1 - \gamma) \log \frac{1 - \gamma}{1 - \epsilon'(n)}.$$

Using Pinsker's inequality [25], for  $n$  large enough, we have that,

$$P(G_6^c) \leq \exp\left(-n\frac{\gamma^2}{2}\right) + e^{-n\tau(1-\rho)}.$$

□

*Lemma 19.* For any  $t$ ,

$$1. \max_{1 \leq u \leq n} F_u(t) \leq K$$

2.

$$\begin{aligned} P(|F(t)|_1 \leq 4n\gamma) &\geq 1 - 2 \exp \left( -n \frac{\gamma^2}{2K^2} \right) \\ &- o(e^{-n}) + 2e^{-bn} \\ &- 2 \exp \left( -nH \left( \frac{q_2^{(C)}}{2} | q_2^{(C)} \right) \right) \\ &- 2 \exp \left( -nH \left( \frac{q_3^{(C)}}{2} | q_3^{(C)} \right) \right) \\ &- 2 \exp \left( -n \frac{\gamma^2}{2} \right) \\ &- 2e^{-n\tau(1-\rho)}. \end{aligned}$$

where  $\gamma$  is as chosen in Lemma 18.

*Proof.* The first part of the lemma follows from the fact that the arrival process for a user  $u$  at the ANs is bounded by  $K$  and therefore, the number of packets for a user  $u$  that cannot be served by the ANs in a given time-slot is  $\leq K$ .

This proof is conditioned on the event  $G_1$  defined in Lemma 17 and  $G_6$  defined in Lemma 18 for time-slots  $t-d$  and  $t-d+1$  and follows on the same lines as that of the proof of Lemma 15.

Conditioned of  $G_1$  and  $G_6$ , we need to compute the probability of the event that not more than  $\frac{\gamma}{K}n$  users each are not connected to the ANs or their locality information is not available in time-slots  $t-1$  and  $t$ .

Let  $G_7$  be the event that more than  $\gamma n$  users' location information is not known in a given time-slot. The probability of this event is  $\epsilon(n)$ , i.i.d. across users and time-slots. Therefore,

$$P(G_7|G_1, G_6) = 2 \exp(-nD(\gamma||\epsilon(n))),$$

where

$$D(\gamma||\epsilon(n)) = \gamma \log \frac{\gamma}{\epsilon(n)} + (1 - \gamma) \log \frac{1 - \gamma}{1 - \epsilon(n)}.$$

Using Pinsker's inequality [25], for  $n$  large enough, we have that,

$$\begin{aligned} P(|F(t)|_1 \leq 4n\gamma) &\geq 1 - \exp\left(-n\frac{\gamma^2}{2K^2}\right) \\ &- P(G_1^c) - P(G_6^c). \end{aligned}$$

Substituting the value of  $P(G_1^c)$  and  $P(G_6^c)$ , the result follows.  $\square$

*Lemma 20.* Let  $\xi(t) := \max_{1 \leq i \leq n} Q_i(t)$  be the maximum queue-length at the end of time-slot  $t$ . Fix a constant

$$\delta \in \left(0, \frac{q_1 \gamma}{K(2 - q_1)}\right).$$

Then,

$$\begin{aligned} P(\xi(t) > \xi(t-1)) &\leq \exp\left(-n\frac{\gamma^2}{2K^2}\right) \\ &+ P(H_3^c) + P(H_4^c) \\ &+ Kn(1 - q_1)^{n\delta} \\ &+ Kn\delta \exp\left(\frac{2n\delta}{q_1} H\left(\frac{q_1}{2} \middle| q_1\right)\right). \end{aligned}$$

*Proof.* Conditioned on the properties of the feedback arrivals from Lemma 19 and Lemma 4 in [16], we have that all feedback arrivals are forwarded directly to the users by the end of  $n$  rounds of channel allocation with probability  $\geq Kn(1 - q_1)^{n\delta} + Kn\delta \exp\left(\frac{2n\delta}{q_1} H\left(\frac{q_1}{2} \middle| q_1\right)\right)$ . This completes the proof.  $\square$

*Lemma 21.* Let  $\xi(t) := \max_{1 \leq i \leq n} Q_i(t)$  be the maximum queue-length at the end of time-slot  $t$ . There exists a constant  $k_0$ , such that

$$P(\xi(t) < \xi(t - k_0)) \geq 1/2.$$

*Proof.* This result follows from Lemma 20 and Lemma 8 in [16].  $\square$

*Proof.* (of Theorem 11) The proof follows from Lemma 20, Lemma 21 and Theorem 5 in [16].  $\square$

## Appendix C

### Proofs from Chapter 4

#### C.1 Proof of Proposition 1

Recall the extended bipartite graph representation discussed in Section 5.3 where we converted the problem of load balancing on  $G(U, V, E)$  to the problem of finding a matching in the bipartite graph  $G_b(U_b, V, E_b)$ .

For any graph  $G$ , on average, the algorithm RANKING proposed in [53] applied on  $G_b(U_b, V, E_b)$  matches the same number of vertices of  $V$  as applying our algorithm on  $G(U, V, E)$ .

Therefore, once the load balancing problem on  $G(U, V, E)$  is converted to a matching problem for the bipartite graph  $G_b(U_b, V, E_b)$ , Proposition A.1.1 follows as a direct consequence of the updated proof of the performance of the algorithm RANKING proposed in [53] for online bipartite matching presented in [13] and [3].

#### C.2 Proof of Theorem 12

To upper bound the performance of all online load balancing algorithms, we construct graph between jobs and servers and specify an arrival sequence on the jobs and upper bound the competitive ratio of any online

load balancing algorithm by upper bounding the performance of any online algorithm for this arrival sequence.

We consider an arrival sequence in which jobs arrive only the first time-slot and all of them have to be served in the next  $b = d_{max}$  time-slots. Therefore, we are looking for an allocation where not more than  $b$  jobs are allocated to any one server. We refer to such an allocation as a  $b$ -Matching.

We visualize the problem of  $b$ -Matching on a bipartite graph between jobs and servers as the problem of finding a matching in a bigger graph where each server is replicated  $b$  times. The analysis in this section uses proof techniques used in [53] to upper bound the performance of all online randomized algorithms for bipartite matching. As in [53], we use a matrix  $S$  to describe the bipartite graph. Let the columns of this matrix represent jobs. Let the set of jobs be  $V$ . Since each server can be matched to  $b$  vertices in  $V$ , we make  $b$  copies of each server in the server set  $U$  to get a set  $U_b$ . The rows of the matrix  $S$  represent vertices in set  $U_b$ . The entry  $S(i, j) = 1$  if  $v_j$  can be served by  $u \in U$  and row  $i$  represents a copy of  $u$  and 0 otherwise. Since each vertex in  $u$  is replicated  $b$  times in  $S$ , the  $b$  rows of  $S$  which represent  $u$  are identical.

The key difference in the analysis in [53] and the analysis for  $b$ -Matching is that the matrix (or graph) used to upper bound the performance of any online algorithm in [53] is the upper triangular matrix which is not a valid matrix in the  $b$ -Matching case as no two rows of the upper triangular matrix are identical. Therefore, several key steps in the proof in [53] do not apply to the  $b$ -Matching problem.



Based on the proof in [53], we now outline our proof in 5 steps.

1. We first characterize a graph  $S$  which we use to upper bound the performance of all online  $b$ -Matching algorithms.
2. We prove that for a graph  $\{S, \pi\}$ , which is obtained by permuting the rows of  $S$  by a permutation  $\pi$ , chosen uniformly at random, the best deterministic algorithm is greedy in the sense that it never leaves a column unmatched if there is an eligible row when this column arrives (Lemma 22).
3. We consider an algorithm called RANDOM [53], which matches each column in  $A$  to a randomly chosen eligible row, independent of all past choices. We show that for an arrival sequence from  $\{S, \pi\}$ , the expected size of matching produced by any deterministic algorithm is the same as the expected size of matching produced by RANDOM on  $S$  (Lemma 24).
4. We use Yao's Lemma [106] to upper bound the expected performance of any randomized algorithm on  $S$  by the expected performance of the best deterministic algorithm when given a problem instance  $\{S, \pi\}$ . Using this and the previous two results, we upper bound the expected performance of any randomized algorithm on  $S$  by the expected performance of RANDOM on  $S$  (Lemma 25).
5. We evaluate the performance of RANDOM on  $S$  to get the upper bound (Lemma 26).

The analysis in [53] also proceeds via Steps 2-5, but for the upper triangular matrix. The specific details of the proofs are different because in our case  $S$  is not upper triangular. The main challenge in our analysis is Step 5. In [53], for the upper triangular matrix, it is possible to exactly characterize the performance of RANDOM. In our case, exact characterization is difficult and therefore, we upper bound the performance of RANDOM on  $S$ . We show that the dominant term in the bound scales correctly and the error term is sublinear in  $|V|$ .

We now characterize the matrix  $S$ . We assume that columns are revealed from right to left. We try to find a matching between  $V$  and  $U_b$  which is equivalent to finding a  $b$ -Matching between  $U$  and  $V$ . We construct the adjacency matrix  $A$  using the  $n \times n$  upper triangular matrix. The first  $b$  columns are identical to the  $n^{th}$  column of the upper triangular matrix, the next  $b$  are identical to the  $(n-1)^{th}$  column and so on. We thus get a set  $V$  such that  $|V| = bn$ . We then replicate each row of  $A$   $b$  times to get a matrix  $S$ . For instance, for  $n = 4$  and  $b = 2$ ,

$$S = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

This point onward, we will focus on algorithms for bipartite matching on  $S$  instead of algorithms for finding  $b$ -Matchings on  $A$ .

For the matrix  $S$  defined above, with every permutation  $\pi$  on  $\{1, \dots, bn\}$  associate a problem instance  $\{S, \pi\}$  such that the adjacency matrix is obtained by permuting the rows of  $S$  under  $\pi$  and the columns or jobs arrive in the order  $bn, bn - 1, \dots, 1$ . Let  $Z$  denote the uniform distribution over the  $(bn)!$  possible permutations.

*Lemma 22.* The best deterministic algorithm is greedy, i.e. it never leaves a column unmatched if there is an eligible row.

*Proof.* Assume that there exists a deterministic algorithm ALG which is not greedy and it is the best possible deterministic algorithm. Now consider an alternative greedy algorithm ALG-GREEDY which does the following. If ALG allocates an incoming column to a particular row, and that row is available then ALG-GREEDY also matches them. If that row is not available, then ALG-GREEDY matches that column to an arbitrary available row. If ALG does not allocate a column to any row, ALG-GREEDY again matches that column to any arbitrary available row.

We will now prove that the number of rows matched by ALG-GREEDY is  $\geq$  the number of rows matched by ALG.

CLAIM: The set of rows matched by ALG-GREEDY at time  $t$  is a superset of the set of rows matched by ALG at time  $t$ .

We prove the claim by induction. It is clearly true for  $t = 1$ . Let it be true at  $t$ . We consider the following three cases:

1. Both ALG and ALG-GREEDY match the next column to the same row.  
In this case, the claim is true for  $t + 1$ .
2. ALG does not match the next column. In this case too, the claim is true for  $t + 1$ .
3. ALG matches the next column to a row  $r$  and ALG-GREEDY does not match it to row  $r$ . This can only happen only if row  $r$  has already been matched by ALG-GREEDY. Therefore the claim is true for  $t + 1$ .

This proves that the number of rows matched produced by ALG GREEDY is  $\geq$  the number of rows matched by ALG.

This contradicts our assumption that the best deterministic algorithm is not greedy, hence the best deterministic algorithm is greedy.  $\square$

*Lemma 23.* For ALG on  $\{S, \pi\}$  and RANDOM on  $\{S, I\}$  where  $I$  is the identity permutation, if the number of eligible rows at time  $t$  is  $k$ , they are equally likely to be any set of  $k$  rows from among the first  $b(n - \lfloor (t - 1)/b \rfloor)$  rows of  $S$ .

*Proof.* We first prove the lemma for RANDOM by induction. The claim is true at time 1. Assume that it is true at time  $t$ .

Case I :  $((t - 1) \bmod b) \neq 0$

By the induction hypothesis at time  $t$ , we have that for the algorithm RANDOM, if there are  $k$  eligible rows, they are equally likely to be any of the first  $b(n - \lfloor (t-1)/b \rfloor)$  rows of  $S$ . The number of eligible rows at time  $t+1$  will then be  $k-1$  since columns  $n-t+1$  and  $n-t$  have the same neighbors. For each subset of eligible rows of size  $k$  at time  $t$ , there are  $k-1$  possible subsets of eligible rows at time  $t+1$  and each one of them is equally likely with probability  $1/k$  because the algorithm RANDOM breaks ties among eligible rows uniformly at random. We therefore have that the claim is true for time  $t+1$  for RANDOM.

Case II :  $((t-1) \bmod b) = 0$

By the induction hypothesis at time  $t$ , we have that for the algorithm RANDOM, if there are  $k$  eligible rows, they are equally likely to be any of the first  $b(n - \lfloor (t-1)/b \rfloor)$  rows of  $S$ . Let  $k'$  of these rows be such that they are eligible to be matched with the column arriving at time  $t+1$ . Since all sets of size  $k$  are equally likely, all sets which contain  $k'$  rows which can be used by the column arriving at  $t+1$  are also equally likely. We consider two cases. In the first case, the column which arrives at  $t$  is matched to one of these  $k'$  rows. The number of eligible rows at time  $t+1$  will then be  $k'-1$ . Since RANDOM breaks ties uniformly at random, this set can be any one of the  $k'-1$  sets with probability  $1/k'$ . The second case is when the column which arrives at  $t$  is not matched to one of these  $k'$  rows. In this case, the number of eligible rows at time  $t+1$  will then be  $k'$  and every such set is equally likely.

The proof of the first claim for ALG also follows by induction on time. It centers on the fact that at any time, if ALG chooses to match column to a particular eligible row, that row is equally likely to be any one of the rows of  $S$  that is eligible to be matched to the column arriving at time  $t$ . This follows from the fact that the permutation  $\pi$  of the rows of  $S$  to get  $\{S, \pi\}$  is chosen uniformly at random from the  $(nb)!$  possible permutations.  $\square$

The next lemma shows that on average, any greedy deterministic algorithm on  $\{S, \pi\}$  has the same performance as that of RANDOM on  $\{S, I\}$ . There is an analogous result in [53] for the upper triangular matrix.

*Lemma 24.* Let ALG be a greedy deterministic online algorithm. The expected size of matching produced by ALG on  $\{S, \pi\}$  where  $\pi$  is picked according to the distribution  $Z$  is the same as the expected size of matching produced by RANDOM on  $(S, I)$ .

*Proof.* We claim that for each  $k$  and  $t$ , the probability that there are  $k$  eligible rows at time  $t$  is the same for RANDOM run on  $(S, I)$  as it is for ALG run on  $\{S, \pi\}$ .

The claim is true for  $t = 1$ . We assume that it is true at time  $t$  and show that this implies that it is also true for time  $t + 1$ .

Case I :  $((t - 1) \bmod b) \neq 0$

Let  $P_{RANDOM}(t, k)$  be the probability that for RANDOM on  $(S, I)$ , there are  $k$  eligible rows at time  $t$  and  $P_{ALG}(t, k)$  the same probability for ALG. By the induction hypothesis, we have that  $P_{RANDOM}(t, k) = P_{ALG}(t, k)$ . Given this, since the columns arriving at  $t$  and  $t+1$  have the same neighbors, we have that  $P_{RANDOM}(t+1, k-1) = P_{RANDOM}(t, k)$  and  $P_{ALG}(t+1, k-1) = P_{ALG}(t, k)$ , and therefore, the claim is true for  $t+1$ .

Case II :  $((t-1) \bmod b) = 0$

By the induction hypothesis we have that  $P_{RANDOM}(t, k) = P_{ALG}(t, k)$ . By the first claim, we have that each such set of  $k$  rows is equally likely for both algorithms. Therefore, the probability that the two sets have  $k'$  rows which can be matched to the vertex arriving at  $t+1$  is also the same. Let this probability be  $\gamma(k', k)$ . Let

$$\beta(k', k) = \left( \gamma(k', k) \cdot \frac{k-k'}{k} + \gamma(k'+1, k) \cdot \frac{k'+1}{k} \right),$$

Since RANDOM breaks ties uniformly at random and ALG breaks ties deterministically, but that particular row is equally likely to be any eligible row  $S$ , we have that

$$P_{RANDOM}(t+1, k') = \sum_{k \geq k'} P_{RANDOM}(t, k) \beta(k', k),$$

and

$$P_{ALG}(t+1, k') = \sum_{k \geq k'} P_{ALG}(t, k) \beta(k', k).$$

Therefore, we conclude that  $P_{RANDOM}(t+1, k') = P_{ALG}(t+1, k')$ .

An incoming column at time  $t$  will not be matched only when there are no rows to match it to. Since  $P_{RANDOM}(t, 0) = P_{ALG}(t, 0)$  for all  $t$ , the result follows.  $\square$

*Lemma 25.* The expected number of rows matched by any online matching algorithm is upper bounded by the expected number of rows matched by RANDOM on  $(S, I)$ .

*Proof.* Fix a randomized algorithm  $R$ . Let  $\mathbb{D}$  be the set of all deterministic algorithms and  $\mathbb{D}_G$  be the set of all greedy deterministic algorithms. Let  $E[R(S, \pi)]$  be the expected size of matching produced by  $R$  on  $(S, \pi)$  and let  $E[D(S, Z)]$  denote the expected size of matching produced by  $D \in \mathbb{D}$  given an input from the distribution  $Z$ . By Yao's Lemma [106], we have that

$$\min_{\pi} \{E[R(S, \pi)]\} \leq \max_{D \in \mathbb{D}} \{E[D(S, Z)]\}.$$

By Lemma 22 we know that the best deterministic algorithm is greedy. Therefore, we can conclude that,

$$\min_{\pi} \{E[R(S, \pi)]\} \leq \max_{D \in \mathbb{D}_G} \{E[D(S, Z)]\}.$$

By Lemma 24 we have the expected size of matching produced by any greedy deterministic algorithm given an instance  $(S, \pi)$  is the same as the expected size of matching produced by RANDOM on  $(S, I)$ . Therefore,

$$\min_{\pi} \{E[R(S, \pi)]\} \leq E[|M_{RANDOM}(S, I)|],$$



where  $M_{RANDOM}(S, I)$  is the matching produced by RANDOM on the input  $(S, I)$ . Therefore, we have that, the performance of any online matching algorithm is upper bounded by the expected size of matching produced by RANDOM on  $(S, I)$ .  $\square$

*Lemma 26.* The expected number of rows matched when RANDOM is executed on  $(S, I)$  is at most  $bn\left(1 - \frac{1}{e}\right) + o(n)$ .

*Proof.* We first classify the columns of the matrix  $S$ . Observe that columns  $bk + 1$  to  $bk + b$  for all integral values of  $k$  such that  $1 \leq k \leq n$  are identical. We refer to columns  $bk + 1$  to  $bk + b$  as columns of type  $k$ . Let  $x(t)$  be the number of types of columns remaining at time  $t$  and  $y(t)$  be the number of rows eligible at time  $t$ . Let  $\Delta x = x(bt + b) - x(bt)$  and  $\Delta y = y(bt + b) - y(bt)$ . For any column  $j$  such that  $bk + 1 \leq j \leq bk + b$ , we refer to the rows  $bk + 1$  to  $bk + b$  as the good rows and the others as bad rows.

The following arguments use the fact we proved in Lemma 23 that if there are  $r$  eligible rows when column  $c$  arrives at time  $t$ , they are equally likely to be any  $r$  of the first  $b(n - \lfloor (t - 1)/b \rfloor)$  rows of  $S$ .

If a column is matched to a bad row by RANDOM when there was at least one good row available, we say that algorithm RANDOM made an error. Let  $E_c$  be the event that there is at least one good row available when column  $c$  comes in. For any column  $c$ ,

$$\begin{aligned} P_c(\text{error}) &= P_c(\text{bad decision}, E_c) \\ &= P_c(\text{bad decision} | E_c) P(E_c). \end{aligned}$$

Let column  $c$  arrive at time  $t$ . By Lemma 23, we have that,

$$P(E_c) = \frac{\binom{bx(t)-1}{y-1}}{\binom{bx(t)}{y}}.$$

Dropping the time index for convenience, we have that,

$$P(E_c) = \frac{y}{x}. \quad (\text{C.1})$$

Additionally we have that

$$P_c(\text{error}|E_c) \geq \frac{y-b}{y}, \quad (\text{C.2})$$

since there can only be at most  $b$  good rows. From Equations C.1 and C.2, we have that,

$$P_c(\text{error}) \geq \frac{y}{x} \frac{y-b}{y} = \frac{y-b}{x}. \quad (\text{C.3})$$

We now bound the expected number of errors made by all columns of a particular type. Let the first column of type  $k$  come in at time  $t$ . Therefore,  $x(t) = k$  and let  $y(t) = y$ . Let  $P_{err}(k)$  be the probability that the first column of type  $k$  makes an error. We claim that

$$P_{err}(k) \geq \frac{y-b}{x}. \quad (\text{C.4})$$

From Equation C.4, we have that,

$$E[\Delta y] \leq -b - b \frac{y-2b}{bx}.$$

Therefore, for the algorithm RANDOM, we have that

$$\frac{E[\Delta y]}{E[\Delta x]} \geq b + \frac{y-b}{x}. \quad (\text{C.5})$$

Consider a system such that

$$\frac{E[\Delta y]}{E[\Delta x]} = b + \frac{y - b}{x}. \quad (\text{C.6})$$

We refer to this system as System B. Next, instead of solving Equation C.6 (a stochastic difference equation), as in [53] we solve Equation C.7 (an ODE). It is known from Kurtz's theorem [56] that the corresponding two solutions tend to each other as  $n \rightarrow \infty$  (and with high probability).

$$\frac{dy}{dx} = b + \frac{y - b}{x}. \quad (\text{C.7})$$

Let  $f(y, x)$  be such that for the original system, System A,

$$\frac{E[\Delta y]}{E[\Delta x]} = f(y, x), \quad (\text{C.8})$$

such that  $f(y, x)$  is an increasing function of  $y$  for  $y > b$ . Note that  $f(y, x)$  is an increasing function of  $y$  for  $y > b$  because the more options an incoming column has, the higher the probability of making an error, since there are always at most  $b$  good rows. From Equation C.5, we know that  $f(y, x) \geq b + \frac{y - b}{x}$ . The solution to System A can be approximated by

$$\frac{dy}{dx} = f(y, x). \quad (\text{C.9})$$

Since both systems start at the same point i.e.  $y = bn, x = n$ , we have that for a given value of  $x$ ,  $y_A(x) \leq y_B(x)$ . We will therefore use System B to get a bound on the performance of System A. We have that

$$\frac{dy}{dx} = b + \frac{y - b}{x}.$$

Solving this, we get that

$$y = cx + bx \log x + b.$$

We know that when  $x = n$ ,  $y = bn$ , therefore, we get that

$$c = b - b \log n + \frac{b}{n}.$$

Therefore,

$$y = bx + bx \log \frac{x}{n} - \frac{bx}{n} + b.$$

At  $y = b$ , we have that

$$x + x \log \frac{x}{n} - \frac{x}{n} = 0,$$

therefore,

$$x = e^{-1+\frac{1}{n}}n.$$

Therefore, the number of columns matched by System B is at most

$$bn - be^{-1+\frac{1}{n}}n + b.$$

Since System A matches fewer columns than System B, we have that the expected size of matching produced by RANDOM is at most

$$bn(1 - e^{-1+\frac{1}{n}} + 1/n).$$

□

*Proof. (Proof of Theorem 1)*

Follows from Lemma 25 and Lemma 26.

□

---

**Algorithm 4** ALG

---

```
1: Create  $d_{max} + t - 1$  copies of each server  $s_i \in S$ .
2: Label the  $j^{th}$  copy of  $s_i$  as  $s_i^{(j)}$ .
3: For each  $s_i^{(j)}$ , pick a value  $V(s_i^{(j)})$  i.i.d. uniform  $[0,1]$ .
4:  $S = \{s_i^{(j)} : s_i \in S, 1 \leq j \leq t + d_{max} - 1\}$ ,  $M = \phi$ .
5: for each time-slot  $u \leq t$  do
6:   for arriving job  $p$  characterized by  $\{S_p, d_p\}$  do
7:      $E(p) = \{s_i \in S_p \text{ and } u \leq j \leq u + d_p - 1\}$ .
8:     if  $(C(p) := E(p) \cap S \setminus M \neq \phi)$  then
9:        $s_{i^*}^{(j^*)} = \arg \min_{x \in C(p)} V(x)$ ,  $M = M \cup s_{i^*}^{(j^*)}$ .
10:    end if
11:  end for
12: end for
```

---

### C.3 Proof of Theorem 13

We propose an algorithm called INSERT RANKING as defined in Section 5.3. In this section, we analyze the performance of this algorithm. Recall the set RANKING defined in Algorithm 1 which is updated after each job allocation and at the end of each time-slot. Each element in RANKING had a value associated with it, which is chosen independently according to the uniform distribution on  $[0, 1]$ .

Consider an arrival sequence  $\mathbb{A}$  which is constructed by appending all arrivals in time-slots 1 to  $t$  such that the relative order of arrival of jobs is maintained.

We define a new algorithm ALG whose performance on  $\mathbb{A}$  is closely related to the performance of INSERT RANKING on all the arrivals from time-slot 1 to  $t$ .

*Lemma 27.* For any arrival process, ALG matches at least  $1 - 1/e$  fraction of the jobs matched by the optimal offline algorithm which knows the entire arrival sequence a-priori.

*Proof.* ALG is identical to the RANKING algorithm proposed in [53]. The result therefore follows from the fact that the competitive ratio of the algorithm RANKING is  $1-1/e$ .  $\square$

For the next Lemma, we focus on arrival sequences such that no jobs arrive after time-slot  $t$ .

*Lemma 28.* For a given sample path (i.e. for the same values of  $V(s_i^j)$  for all  $s_i \in S$  and  $1 \leq j \leq t + d_{max} - 1$ ), for a given arrival process, the set of jobs matched by ALG is a subset of the set of jobs matched by INSERT RANKING.

*Proof.* Recall Step 15 of Algorithm 1 where matched server copies re-inserted into the set of unmatched servers.

For a job  $p$ , the sets of eligible server copies that it can be matched to are denoted by  $N(p)$  for INSERT RANKING and  $C(p)$  for ALG.

We claim the following properties:

P1: When a job  $p$  arrives, the set of jobs that arrived before  $p$  and were matched by INSERT RANKING is a superset of the the set of jobs that arrived before  $p$  and were matched by ALG.

P2: When a job  $p$  arrives in time-slot  $t$ , the set unmatched server copies is a superset of server copies with indices  $t \leq j \leq t + d_{max} - 1$  not matched by ALG.

The two algorithms are identical in the first time-slot. Consider the first arrival  $p$  in the second time-slot. Consider the sets of eligible server copies that it can be matched to ( $N(p)$  for INSERT RANKING and  $C(p)$  for ALG). Due to reinsertion,  $N(p) \supseteq C(p)$ . There are three possible cases:

- I: INSERT RANKING matches  $p$  to a reinserted server copy. In this case, P1 and P2 follow directly.
- II: INSERT RANKING matches  $p$  to a non-reinserted server copy. As before, P1 follows. If ALG also matches  $p$  to the same server copy, P2 also follows. If not, since  $N(p) \supseteq C(p)$ , INSERT RANKING matches  $p$  to a non-reinserted server copy  $s_i^{(j)} \in \text{RANKING}$  with a lower value. Since ALG does not match  $p$  to this server, we conclude that  $s_i^{(j)}$  has already been matched by ALG to some other job and therefore P2 follows.
- III: INSERT RANKING does not match  $p$  to any server copy. Since  $N(p) \supseteq C(p)$ , this implies ALG also does not match  $p$  to any server copy and P1 and P2 follow.

By induction, the proof follows for all subsequent arrivals. By P1, we conclude that the set of jobs matched by ALG is a subset of the set of jobs matched by INSERT RANKING.  $\square$

*Proof. (Proof of Theorem 13)* We first bound the quantity  $\rho_t(\text{INSERT RANKING})$ .

By Lemmas 27 and 28, we conclude that

$$\rho_t(\text{INSERT RANKING}) \geq 1 - \frac{1}{e},$$

and therefore,

$$\begin{aligned} & \rho(\text{INSERT RANKING}) \\ &= \inf_t \rho_t(\text{INSERT RANKING}) \geq 1 - \frac{1}{e}. \end{aligned}$$

□

## C.4 Alternative Algorithms: Proofs of Theorems 14 and 15

From Sections C.2 and C.3 we conclude that INSERT RANKING is an optimal online algorithm. In this section, we try to understand if correlated randomness is necessary to achieve optimality or if just randomization is sufficient.

### C.4.1 RANDOMIZED JSQ

The first algorithm which we study is a Join the Shortest Queue algorithm which break ties between the shortest queues uniformly at random independent of all past choices. We refer to this algorithm as RANDOMIZED JSQ. The algorithm has been defined in Section 5.3.

*Proof. (Proof of Theorem 14)*

To compute an upper bound on the competitive ratio of RANDOMIZED JSQ,



we construct a bad arrival sequence and upper bound the performance of RANDOMIZED JSQ by its performance for that particular arrival sequence. We construct the arrival sequence as follows. Consider a matrix such that  $A(i, i) = 1$  for  $1 \leq i \leq n$ ,  $A(i, j) = 1$  if  $j \geq n/2$ ,  $i \leq n/2$  and  $A(i, j) = 0$  otherwise. As in Section C.2, columns of  $A$  represent jobs and rows represent servers. The jobs (columns) arrive from right to left. For example, for  $n = 8$ , this matrix is

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We set the deadline of all jobs to be 1. Consider an arrival process  $Arr$ , where  $Arr(t) = A$  for all  $t$ . At time  $t$ , all jobs that arrived before time-slot  $t$  are no longer in the system. They are either served or dropped. Since each job has a service time of 1 time-slot, for this arrival process, the task of load balancing at time  $t$  is equivalent to the task of finding an online matching. The proof is based the analysis for a similar arrival sequence presented in [69]. However, as the arrival sequence is different, the proof is structurally similar, but the detailed analysis differs.

We focus on the first  $n/2$  servers. Let  $x(c)$  and  $y(c)$  be random variables denoting the number of jobs remaining and the number of servers among the first  $n/2$  servers which have not been matched by the time the  $(n - c + 1)^{th}$

column arrives. Let

$$\Delta x = x(c) - x(c+1),$$

$$\Delta y = y(c) - y(c+1).$$

$$\Delta y = -1 \text{ w.p. } \frac{y}{y+1}$$

and 0 otherwise. Therefore, using Kurtz's Theorem [56], we approximate this system by the system given by

$$\frac{dy}{dx} = \frac{y}{y+1}.$$

Solving for  $y$  with initial conditions  $x = n$ ,  $y = n/2$ , we get that:

$$x = y + \frac{n}{2} + \log \frac{2y}{n}.$$

For  $x = n/2$ , we have that:

$$y + \log \frac{2y}{n} = 0.$$

Therefore,  $y(n/2) < \log n$ . Therefore, we have that,

$$\rho_t(\text{RANDOMIZED JSQ}) \leq \frac{t(n/2 + \log n)}{tn}.$$

As  $n \rightarrow \infty$ ,

$$\rho_t(\text{RANDOMIZED JSQ}) \leq \frac{1}{2}.$$

Therefore, we have that

$$\rho(\text{RANDOMIZED JSQ}) \leq \frac{1}{2}.$$

□

### C.4.2 RANDOMIZED P-JSQ

Since INSERT RANKING is not a join the short queue algorithm, but, is biased towards joining shorter queues, we analyze the performance of an algorithm which we call RANDOMIZED P-JSQ. Like INSERT RANKING, this algorithm is also biased towards joining shorter queues. The algorithm has been defined in Section 5.3.

*Proof. (Proof of Theorem 15)*

To compute an upper bound on the competitive ratio of RANDOMIZED P-JSQ, we construct a bad arrival sequence and upper bound the performance of RANDOMIZED P-JSQ by its performance for that particular arrival sequence. We construct the arrival sequence as follows.

Construct a matrix  $B$  such that  $B(i, i) = 1$  and  $B(i, j) = 1$  for  $i \leq n/2$  and  $j > n/2$ . For example for  $n = 4$ ,

$$B = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We then make  $d_{max}$  copies for each column to get the arrival matrix  $A$ . For  $d_{max} = 2$ ,

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

We set the deadline of all jobs to be  $d_{max}$ . Like in Section C.2, consider an arrival process  $Arr$ , where  $Arr(1) = A$  and  $Arr(t) = \phi$  for  $t \neq 1$ . We construct

a matrix  $S$  from  $A$  by creating  $d_{max}$  copies of each server and try to find a matching for  $S$ . In this case where  $d_{max} = 2$ ,

$$S = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

We first classify the columns of the matrix  $S$ . Observe that columns  $d_{max}k + 1$  to  $d_{max}k + d_{max}$  for all integral values of  $k$  such that  $1 \leq k \leq n$  are identical. We refer to columns  $d_{max}k + 1$  to  $d_{max}k + d_{max}$  as columns of type  $k$ . Let  $x(t)$  be the number of types of columns remaining at time  $t$  and  $y(t)$  be the number of the top  $d_{max}n/2$  rows eligible at time  $t$ . Let  $\Delta x = x(d_{max}t + d_{max}) - x(d_{max}t) = -1$  and  $\Delta y = y(d_{max}t + d_{max}) - y(d_{max}t)$ . For any column  $j$  such that  $d_{max}k + 1 \leq j \leq d_{max}k + d_{max}$ , we refer to the rows  $d_{max}k + 1$  to  $d_{max}k + d_{max}$  as the good rows and the others as bad rows.

Let the first column of type  $k$  come in at time  $t$ . The quantity  $y(t)$  decreases by at most 1 after each column gets matched since all columns of type  $k$  have the same set of eligible rows. Therefore,  $y > y(t) - d_{max}$  for all columns of type  $k$ . For any column of type  $k$ , the probability of making an error is at least  $\frac{y}{y + d_{max}} \geq \frac{y(t) - d_{max}}{y(t) + d_{max}}$ . Therefore,

$$E[\Delta y] \leq -d_{max} \frac{y(t) - d_{max}}{y(t) + d_{max}}.$$

Since  $\Delta x = -1$ , we have that

$$\frac{E[\Delta y]}{E[\Delta x]} \geq d_{max} \frac{y(t) - d_{max}}{y(t) + d_{max}}.$$

Instead of solving the stochastic difference equation, we use Kurtz's theorem [56] and bound its solution by the solution to

$$\frac{dy}{dx} = d_{max} \frac{y - d_{max}}{y + d_{max}},$$

with the initial condition  $x = n$ ,  $y = d_{max}n/2$ . We then compute the value of  $y$  for  $x = n/2$ . We get that when  $x = n/2$ ,  $y \leq 2d_{max} \log n$ . Therefore, we have that,

$$\rho_t(\text{RANDOMIZED P-JSQ}) \leq \frac{d_{max}n/2 + 2d_{max} \log n}{d_{max}n}.$$

As  $n \rightarrow \infty$ ,

$$\rho_t(\text{RANDOMIZED P-JSQ}) \leq \frac{1}{2}.$$

Therefore, we have that

$$\rho(\text{RANDOMIZED P-JSQ}) \leq \frac{1}{2}.$$

□

## Appendix D

### Proofs from Chapter 5

#### D.1 Proof of Theorem 16

We first present an outline of the proof of Theorem 16. We consider two cases. We first focus on the case when the learning-based storage policies use fewer than  $n$  arrivals to learn the distribution.

1. If the learning phase lasts for the first  $n^\gamma$  arrivals for some  $0 < \gamma \leq 1$ , we show that under Assumption (i), w.h.p., in the learning phase, there are no arrivals for at least  $n - O(n^{\frac{\gamma}{\beta}})$  content types. (Lemma 29).
2. Next, we show that w.h.p., among the first  $n^\gamma$  arrivals, i.e., during the learning phase,  $\Omega(n^\gamma)$  requests are deferred (Lemma 31).
3. Using Lemma 29, we compute a lower bound on the number of requests deferred in Phase 2 (after the learning phase) by any learning-based static storage policy (Lemma 32).
4. Using Steps 2 and 3, we lower bound the number of requests deferred in the interval of interest.

In the case when the learning phase lasts for more than  $n$  arrivals, we show that the number of requests deferred in the learning phase alone is  $\Omega(n)$ ,

thus proving the theorem for this case.

*Lemma 29.* Let  $E_1$  be the event that in the first  $n^\gamma$  arrivals, for  $0 < \gamma < 1$  no more than  $O(n^{\frac{\gamma}{\beta}})$  different types of contents are requested. Then,

$$\mathbb{P}(E_1^c) = o\left(\frac{1}{n}\right). \quad (\text{D.1})$$

for  $n$  large enough.

*Proof.* Recall  $\lambda_i = \bar{\lambda} n p_i$  where  $p_i = \frac{i^{-\beta}}{Z(\beta)}$  for  $Z(\beta) = \sum_{i=1}^m i^{-\beta}$ .

$$Z(\beta) = \sum_{i=1}^{\alpha n} i^{-\beta} \geq \int_1^{\alpha n+1} i^{-\beta} di \geq \frac{0.9}{\beta-1}$$

for  $n$  large enough. Therefore, for all  $i$ ,

$$p_i \leq \frac{\beta-1}{0.9} i^{-\beta}.$$

The total mass of all content types  $i = k, ..m = \alpha n$  is

$$\sum_{i=k}^{\alpha n} p_i \leq \sum_{i=k}^{\alpha n} \frac{\beta-1}{0.9} i^{-\beta} \leq \int_{k-1}^{\alpha n} \frac{\beta-1}{0.9} i^{-\beta} di \leq \frac{1}{0.9} \frac{1}{(k-1)^{\beta-1}}.$$

Now, for  $k = (n)^{\gamma/\beta} + 1$ , we have that,

$$\sum_{i=k}^{\alpha n} p_i \leq \frac{1}{0.9} \frac{n^{\gamma/\beta}}{n^\gamma}.$$

Therefore, the expected number of requests for content types  $k, k+1, ..\alpha n$  is less than  $\frac{1}{0.9}(n^{\gamma/\beta})$ . Using the Chernoff bound, the probability that there are more than  $\frac{2}{0.9}(n^{\gamma/\beta})$  requests for content types  $k, k+1, ..\alpha n$  in the interval of interest is less than  $\frac{1}{n^2}$  for  $n$  large enough.

Therefore, with probability greater than  $1 - 1/n^2$ , the number different types of contents requests for in the interval of interest is less than  $n^{\gamma/\beta} + \frac{2}{0.9}(n^{\gamma/\beta})$ . Hence the result follows.  $\square$

We use the following concentration result for Exponential random variables.

*Lemma 30.* Let  $X_k$  for  $0 \leq k \leq v$ , be i.i.d. exponential random variables with mean 1, then,

$$\mathbb{P}\left(\sum_{k=1}^v X_i \leq a\right) \leq \exp(v - a) \left(\frac{a}{v}\right)^v. \quad (\text{D.2})$$

*Proof.* This follows from elementary calculations, and is provided here for completeness. For any  $a$  and  $v$ , by the Chernoff bound, we have that,

$$\mathbb{P}\left(\sum_{k=1}^v X_i \leq a\right) \leq \min_{t>0} e^{ta} (E[e^{-tX_i}])^v.$$

Since  $X_k$  is an exponential random variable with mean 1, we have that,

$$\mathbb{P}\left(\sum_{k=1}^v X_i \leq a\right) \leq \min_{t>0} e^{ta} \left(\frac{1}{1+t}\right)^v = \exp(v - a) \left(\frac{a}{v}\right)^v.$$

$\square$

*Lemma 31.* Suppose the system starts with each content piece stored on exactly one server. Let  $E_2$  be the event that in the first  $n^\gamma$  arrivals for  $\gamma$  such that  $0 < \gamma < 1$ , at most  $O(n^{\gamma/\beta})(\log n + 1)$  are served (not deferred). Then, for  $\beta > 1$ ,

$$\mathbb{P}(E_2) \geq 1 - \frac{1}{\log n}. \quad (\text{D.3})$$



*Proof.* This proof is conditioned on the event  $E_1$  defined in Lemma 29. Conditioned on  $E_1$ , in the first  $n^\gamma$  arrivals, at most  $O(n^{\gamma/\beta})$  different content types are requested. Therefore, at most  $O(n^{\gamma/\beta})$  servers can serve requests during the first  $n^\gamma$  arrivals.

Let  $E_3$  be the event that the time taken for the first  $n^\gamma$  arrivals is less than  $\frac{2n^\gamma}{\lambda n}$ . Since the expected time for the first  $n^\gamma$  arrivals is  $\frac{n^\gamma}{\lambda n}$ , by the Chernoff bound,  $\mathbb{P}(E_3) \geq 1 - o(1/n)$ . The rest of this proof is conditioned on the event  $E_3$ .

If the system serves (does not defer) more than  $O(n^{\gamma/\beta}(\log n + 1))$  requests in this interval, at least one server needs to serve more than  $\log n$  requests. By substituting  $a = cn^{-1+\gamma}$  and  $v = \log n$  in Lemma 30, we have that,

$$\begin{aligned} \mathbb{P}\left(\sum_{k=1}^{\log n} X_k \leq cn^{-1+\gamma}\right) &\leq \exp(\log n - cn^{-1+\gamma}) \\ &\quad \times \left(\frac{cn^{-1+\gamma}}{\log n}\right)^{\log n} = o\left(\frac{1}{n}\right). \end{aligned}$$

Therefore, the probability that a server serves more than  $\log n$  requests in an interval of  $\frac{2n^\gamma}{\lambda n}$  time is  $o(\frac{1}{n})$ . Therefore, using the union bound, the probability that none of these  $O(n^{\gamma/\beta})$  servers serve more than  $\log n$  requests each in  $\frac{2n^\gamma}{\lambda n}$  time is greater than  $1 - O(n^{\gamma/\beta})o(\frac{1}{n})$ . Therefore, we have that,

$$\begin{aligned} \mathbb{P}(E_2^c) &\leq O(n^{\gamma/\beta})o\left(\frac{1}{n}\right) + P(E_1^c) + P(E_3^c) \\ &\leq \frac{1}{\log n} \end{aligned}$$

for  $n$  large enough. □

*Lemma 32.* Let the interval of interest be  $T(n)$  such that  $T(n) = \Omega(1)$ . If the learning phase of the storage policy lasts for the first  $n^\gamma$  arrivals,  $0 < \gamma < 1$ , the expected number of requests deferred in Phase 2 is  $\Omega(T(n)n^{1-\gamma}n^{\frac{\gamma}{\beta}})$ .

*Proof.* Let  $N_2$  be the number of arrivals in Phase 2, then we have that,  $E[N_2] = T(n)\bar{\lambda}n - n^\gamma$ .

Let  $E_4$  be the event that  $N_2 > E[N_2]/2$ . Using the Chernoff bound, it can be shown that  $P(E_4^c) = o(1/n)$ .

The rest of this proof is conditioned on  $E_1$  defined in Lemma 29 and  $E_4$  defined above. We consider the following two cases depending on the number of servers allocated to content types not seen in Phase 1.

Case I: The number of servers allocated to content types not seen in Phase 1 is less than  $\epsilon n$  for some  $\epsilon \leq 1 - \frac{\bar{\lambda}}{1000}$ . For  $\beta > 1$ ,

$$Z(\beta) = \sum_{i=1}^{\alpha n} i^{-\beta} \leq \sum_{i=1}^{\infty} i^{-\beta} = c_z < \infty.$$

Therefore, for all  $i$ ,  $p_i \geq \frac{1}{c_z} i^{-\beta}$ . The total mass of all content types  $k, k+1, \dots, \alpha n$  is

$$\begin{aligned} \sum_{i=k}^{\alpha n} p_i &\geq \sum_{i=k}^{\alpha n} \frac{1}{c_z} i^{-\beta} \geq \int_k^{\alpha n+1} \frac{1}{c_z} i^{-\beta} di \\ &= \frac{0.9}{c_z(\beta-1)} \frac{1}{k^{\beta-1}}, \end{aligned}$$

for  $n$  large enough.

Therefore, the expected number of arrivals of types not requested in Phase 1 in Phase 2 is at least  $(\frac{T(n)\bar{\lambda}n - n^\gamma}{2}) \frac{0.9}{c_z(\beta-1)} \frac{n^{\frac{\gamma}{\beta}}}{n^\gamma}$ .

Let  $E_5$  be the event that in Phase 2, there are at least  $(\frac{T(n)\bar{\lambda}n - n^\gamma}{4}) \frac{0.9}{c_z(\beta-1)} \frac{n^{\frac{\gamma}{\beta}}}{n^\gamma}$  arrivals of types not requested in Phase 1. Using the Chernoff bound,  $\mathbb{P}(E_5^c) = o(1/n)$ .

Conditioned on  $E_1$ , all but  $O(n^{\gamma/\beta})$  content types, are not requested in Phase 1. Recall that all learning-based policies treat all these content types equally and that the total number of servers allocated to store the content types not seen in Phase 1 is less than  $\epsilon n$ . Let  $\eta$  be the probability that a content is not stored by the storage policy under consideration. Then,

$$\eta \geq 1 - \frac{\epsilon n}{n - O(n^{\gamma/\beta})} \geq 1 - \frac{\epsilon}{2},$$

for  $n$  large enough.

Let  $E_6 = E_1 \cap E_3 \cap E_4 \cap E_5$  and  $D_2$  be the number of requests deferred in Phase 2.

$$\begin{aligned} E[D_2|E_6] &\geq \eta \left( \left( \frac{T(n)\bar{\lambda}n - n^\gamma}{2} \right) \frac{0.9}{2c_z(\beta-1)} \frac{n^{\gamma/\beta}}{n^\gamma} \right) \\ &\geq \left( 1 - \frac{\epsilon}{2} \right) \left( \frac{T(n)\bar{\lambda}n - n^\gamma}{2} \right) \frac{0.9}{2c_z(\beta-1)} \frac{n^{\gamma/\beta}}{n^\gamma} \\ &= \Omega(T(n)n^{1-\gamma}n^{\gamma/\beta}). \end{aligned}$$

Therefore,

$$\begin{aligned}
E[D_2] &\geq E[D_2|E_6]\mathbb{P}(E_6) \\
&\geq E[D_2|E_6]\left(1 - \frac{1}{\log n} - \frac{3}{n}\right) \\
&= \Omega(T(n)n^{1-\gamma}n^{\gamma/\beta}).
\end{aligned}$$

Case II: The number of servers allocated to content types not seen in Phase 1 is more than  $\epsilon n$  for some  $\epsilon > 1 - \frac{\bar{\lambda}}{1000}$ .

Let  $f(n)$  be the number of servers allocated to store all content types that are requested in Phase 1. By our assumption,  $f(n) \leq \frac{\bar{\lambda}}{1000}n$ .

Let  $\mathbf{C}_1$  be the set of content types requested in Phase 1. Let  $p = \sum_{c \in \mathbf{C}_1} p_c$  be the total mass of all content types  $c \in \mathbf{C}_1$ . Let  $\hat{p}_c$  be the fraction of requests for content-type  $c$  in Phase 1. By the definition of  $\mathbf{C}_1$ , the total empirical mass of all content types  $c \in \mathbf{C}_1$  is obviously  $\hat{p} = \sum_{c \in \mathbf{C}_1} \hat{p}_c = 1$ .

Recall that there are  $n^\gamma$  arrivals in Phase 1. Let  $r = n^\gamma$ . We now use the Chernoff bound to compute a lower bound on the true mass  $p$ , using a technique similar to that used in [66] (Lemma 4). By the Chernoff bound, we know that,

$$\mathbb{P}(\hat{p} > (1 + \kappa)p) \leq \exp\left(-\frac{pr\kappa^2}{3}\right).$$

Let  $\delta = \exp\left(-\frac{pr\kappa^2}{3}\right)$ , then, we have that, with probability greater than  $1 - \delta$ ,

$$\hat{p} - p > \sqrt{\frac{-3p \log \delta}{r}}.$$

Solving for  $p$ , we get that, with probability greater than  $1 - \delta$ ,  $p > 1 - \frac{3 \log(1/\delta)}{2r}$ , for  $n$  large enough. Let  $\delta = 1/n$ , then we have that, with probability greater than  $1 - 1/n$ ,  $p > 1 - \frac{3 \log n}{2n^\gamma}$ . Conditioned on the event  $E_4$ , there are at least  $\frac{T(n)\bar{\lambda}n - n^\gamma}{2}$  arrivals in Phase 2. The remainder of this proof is conditioned on  $E_4$ . Let  $A_2$  be the number of arrivals of types  $c \in \mathbf{C}_1$  in phase 2. Let  $E_7$  be the event that

$$A_2 > \frac{T(n)\bar{\lambda}n - n^\gamma}{2} \left(1 - \frac{3 \log n}{2n^\gamma}\right).$$

Since the expected number of arrivals of content types  $c \in \mathbf{C}_1$  in Phase 2 is at least

$$(T(n)\bar{\lambda}n - n^\gamma) \left(1 - \frac{3 \log n}{2n^\gamma}\right),$$

using the Chernoff bound, we can show that  $\mathbb{P}(E_7^c) = o(1/n)$ . The rest of this proof is conditioned on  $E_7$ . By our assumption, the number of servers which can serve arrivals of types  $c \in \mathbf{C}_1$  in Phase 2 is  $f(n)$ . Therefore, if at least  $A_2/2$  requests are to be served in Phase 2, the sum of the service times of these  $A_2/2$  requests should be less than  $T(n)f(n)$  (since the number of servers which can serve these requests is  $f(n)$ ). Let  $E_8$  be the event that the sum of  $A_2/2$  independent Exponential random variables with mean 1 is less than  $T(n)f(n)$ . By substituting  $v = A_2/2$  and  $a = T(n)f(n)$  in Lemma 30, we have that,

$$\begin{aligned} \mathbb{P}(E_8) &\leq \exp\left(\frac{A_2}{2} - T(n)\right) \left(\frac{2T(n)f(n)}{A_2}\right)^{\frac{A_2}{2}} \\ &\leq \exp\left(\frac{A_2}{2}\right) \left(\frac{2T(n)f(n)}{A_2}\right)^{\frac{A_2}{2}} = o\left(\frac{1}{n}\right) \end{aligned}$$

for  $n$  large enough. Hence,

$$\begin{aligned}\mathbb{P}\left(D_2 \geq \frac{A_2}{2}\right) &\geq 1 - \mathbb{P}(E_1^c) - o\left(\frac{1}{n}\right) \\ \Rightarrow E[D_2] &= \Omega(T(n)n^{1-\gamma}n^{\gamma/\beta}).\end{aligned}$$

□

*Proof.* (Proof of Theorem 16)

We consider two cases:

Case I: The learning phase lasts for the first  $n^\gamma$  arrivals where  $0 \leq \gamma < 1$ .

Let  $D_1$  be the number of requests deferred in Phase 1 and  $D$  be total number of requests deferred in the interval of interest. Then, we have that,

$$E[D] = E[D_1] + E[D_2].$$

By Lemmas 31 and 32 and since  $T(n) = \Omega(1)$ , we have that,

$$\begin{aligned}E[D] &\geq n^\gamma - (n^\gamma \log n)^{\frac{1}{\beta-1}} \log n + E[D_2] \\ &= \Omega(nT(n))^{\frac{1}{2-1/\beta}}.\end{aligned}$$

Case II: The learning phase lasts for longer than the time taken for the first  $n$  arrivals.

By Lemma 31, the number of requests deferred in the first  $n$  arrivals is at least  $n - O(n^{1/\beta} \log n)$  with probability greater than  $1 - 1/\log n$ . Therefore, we have that,

$$\begin{aligned}E[D] &\geq \left(n - O(n^{1/\beta} \log n)\right) \left(1 - \frac{1}{\log n}\right) = \Omega(n) \\ &= \Omega(nT(n))^{\frac{1}{2-1/\beta}}.\end{aligned}$$

□

## D.2 Proof of Theorem 17

In this section, we provide an outline of the proof of Theorem 17. The proof follows on the same lines as the proof of Theorem 16.

1. First, we show that w.h.p., among the first  $n^\gamma$  arrivals, i.e., during the learning phase,  $\Omega(n^\gamma)$  requests are deferred (Lemma 31).
2. Since we are studying the performance of the MYOPIC policy for the Continuous Change Model, the relative order of popularity of contents keeps changing in the interval of interest. If the learning phase lasts for the first  $n^\gamma$  arrivals for some  $0 < \gamma \leq 1$ , we show that under Assumption (i), w.h.p., in the learning phase, only  $O(n^{\gamma/\beta})$  content types are requested.
3. Next, we show that the expected the number of requests in Phase 2 for content types not requested in Phase 1 is  $\Omega(n^{1-\gamma}n^{\gamma/\beta})$ . Using this, we compute a lower bound on the number of requests deferred in Phase 2 (after the learning phase) by any learning-based static storage policy. This results follows by the same arguments as the proof of Lemma 32.
4. Using Steps 1 and 3, we lower bound the number of requests deferred in the interval of interest.

## D.3 Proof of Theorem 18

We first present an outline the proof of Theorem 18.

1. We first show that under Assumption (i), on every arrival in the interval of interest  $(T(n))$ , there are  $\Theta(n)$  idle servers w.h.p. (Lemma 34).
2. Next, we show that w.h.p., in the interval of interest of length  $T(n)$ , only  $O((nT(n))^{\frac{1}{\beta}})$  unique content types are requested (Lemma 35).
3. Conditioned on Steps 1 and 2, we show that, the MYOPIC policy ensures that in the interval of interest, once a content type is requested for the first time, there is always at least one idle server which can serve an incoming request for that content.
4. Using Step 3, we conclude that, in the interval of interest, only the first request for a particular content type will be deferred. The proof of Theorem 18 then follows from Step 2.

*Lemma 33.* Let the cumulative arrival process to the content delivery system be a Poisson process with rate  $\bar{\lambda}n$ . At time  $t$ , let  $\chi(t)$  be the number of occupied servers under the MYOPIC storage policy. Then, we have that,  $\chi(t) \leq_{st} S(t)$ , where  $S(t)$  is a poisson random variable with rate  $\bar{\lambda}n(1 - e^{-t})$ .

*Proof.* Consider an  $M/M/\infty$  queue where the arrival process is Poisson( $\bar{\lambda}n$ ). Let  $S(t)$  be the number of occupied servers at time  $t$  in this system. It is well known that  $S(t)$  is a Poisson random variable with rate  $\bar{\lambda}n(1 - e^{-t})$ . Here we provide a proof of this result for completeness. Consider a request  $r^*$  which arrived into the system at time  $t_0 < t$ . If the request is still being served by a



server, we have that,

$$t_0 + \mu(r^*) > t,$$

where  $\mu(r^*)$  is the service time of request  $r^*$ . Since  $\mu(r^*) \sim \text{Exp}(1)$ , we have that,

$$\mathbb{P}(\mu(r^*) > t - t_0 | t_0) = e^{-(t-t_0)}.$$

Therefore,

$$\begin{aligned} \mathbb{P}(r^* \text{ in the system at time } t) &\leq \int_0^t \frac{1}{t} e^{-(t-t_0)} dt_0 \\ &= \frac{1 - e^{-t}}{t}. \end{aligned}$$

Therefore, every request that arrived in the system is still in the system with probability at most  $\frac{1-e^{-t}}{t}$ . Since the arrival process is Poisson, the number of requests in the system at time  $t$  is stochastically dominated by a Poisson random variable with rate  $\bar{\lambda}nt\left(\frac{1-e^{-t}}{t}\right) = \bar{\lambda}n(1 - e^{-t})$ .

To show  $\chi(t) \leq_{st} S(t)$ , we use a coupled construction similar to Figure D.1. The intuition behind the proof is the following: the rate of arrivals to the content delivery system and the  $M/M/\infty$  system (where each server can serve all types of requests) is the same. The content delivery system serves fewer requests than the  $M/M/\infty$  system because some requests are deferred even when the servers are idle. Hence, the number of busy servers in the content delivery system is stochastically dominated by the number of busy servers in the  $M/M/\infty$  queueing system.  $\square$

*Lemma 34.* Let the interval of interest be  $[t_0, t_0 + T(n)]$  where  $T(n) = o(n^{\beta-1})$  and  $\varepsilon \leq \frac{1-\bar{\lambda}}{2}$ . Let  $F_1$  be the event that at the instant of each arrival in the interval of interest, the number of idle servers in the system is at least  $(1 - \bar{\lambda} - \varepsilon)n$ . Then,  $\mathbb{P}(F_1^c) = o(\frac{1}{n})$ .

*Proof.* Let  $F_2$  be the event that the number of arrivals in  $[t_0, t_0 + T(n)] \leq nT(n)(\bar{\lambda} + \varepsilon)$ . Using the Chernoff bound for the Poisson process, we have that,

$$\mathbb{P}(F_2^c) = o\left(\frac{1}{n}\right).$$

Consider any  $t \in [t_0, t_0 + T(n)]$ . By Lemma 33,  $\chi(t) \leq_{st} S(t)$ , where  $S(t) \sim \text{Poisson}(\bar{\lambda}n(1 - e^{-t}))$ . Therefore,

$$\mathbb{P}(\chi(t) > (\bar{\lambda} + \varepsilon)n) \leq \mathbb{P}(S(t) > (\bar{\lambda} + \varepsilon)n).$$

Moreover,  $S(t) \leq_{st} W(t)$  where  $W(t) = \text{Poisson}(\bar{\lambda}n)$ . Therefore, using the Chernoff bound for  $W(t)$ , we have that,

$$\mathbb{P}(S(t) > (\bar{\lambda} + \varepsilon)n) \leq \mathbb{P}(W(t) > (\bar{\lambda} + \varepsilon)n) = e^{-c_1 n},$$

for some constant  $c_1 > 0$ . Therefore,

$$\begin{aligned} \mathbb{P}(F_1^c) &\leq \mathbb{P}(F_2^c) + (\bar{\lambda} + \varepsilon)nT(n)\mathbb{P}(\chi(t) > (\bar{\lambda} + \varepsilon)n) \\ &= o\left(\frac{1}{n}\right). \end{aligned}$$

□

*Lemma 35.* Let  $F_3$  be the event that in the interval of interest of duration  $T(n)$  such that  $T(n) = o(n^{\beta-1})$ , no more than  $O((nT(n))^{1/\beta})$  different types of contents are requested. Then,  $\mathbb{P}(F_3^c) = o(\frac{1}{n})$ .

*Proof.* Recall from the proof of Lemma 29 that the total mass of all content types  $k, \dots, m = \alpha n$  is

$$\sum_{i=k}^{\alpha n} p_i \leq \frac{1}{0.9} \frac{1}{(k-1)^{\beta-1}}.$$

Now, for  $k = (nT(n))^{1/\beta} + 1$ , we have that,

$$\sum_{i=k}^{\alpha n} p_i \leq \frac{1}{0.9} (nT(n))^{-\frac{\beta-1}{\beta}}.$$

Conditioned on the event  $F_2$  defined in Lemma 34, the expected number of requests for content types  $k, k+1, \dots, \alpha n$  is less than  $\frac{1}{0.9}(\bar{\lambda} + \varepsilon)(nT(n))^{1/\beta}$ . Using the Chernoff bound, the probability that there are more than  $\frac{2}{0.9}(\bar{\lambda} + \varepsilon)(nT(n))^{1/\beta}$  requests for content types  $k, k+1, \dots, \alpha n$  in the interval of interest is less than  $\frac{1}{n^2}$  for  $n$  large enough.

Therefore, with probability greater than  $1 - 1/n^2 - \mathbb{P}(F_2^c)$ , the number different types of contents requests for in the interval of interest is less than  $(nT(n))^{1/\beta} + \frac{2}{0.9}(\bar{\lambda} + \varepsilon)(nT(n))^{1/\beta}$ . Hence the result follows.  $\square$

*Proof.* (Proof of Theorem 18)

Let  $F_4$  be the event that, in the interval of interest, every request for a particular content type except the first request is not deferred. The rest of this

proof is conditioned on  $F_1$  and  $F_3$ . Let  $U(t)$  be the number of unique contents which have been requested in the interval of interest before time  $t$  for  $t \in [t_0, t_0 + T(n)]$ . Conditioned on  $F_3$ , as defined in Lemma 35,  $U(t) \leq k_1(nT(n))^{1/\beta}$  for some constant  $k_1 > 0$  and  $n$  large enough. Conditioned on  $F_1$ , there are always  $(1 - \bar{\lambda} - \varepsilon)n$  idle servers in the interval of interest.

CLAIM: For every  $i$  and  $n$  large enough, once a content  $C_i$  is requested for the first time in the interval of interest, the MYOPIC policy ensures that there is always at least 1 idle server which can serve a request for  $C_i$ .

Note that since  $T(n) = o(n^{\beta-1})$ ,  $(nT(n))^{1/\beta} = o(n)$ . Let  $n$  be large enough such that  $k_1(nT(n))^{1/\beta} < (1 - \bar{\lambda} - \varepsilon)n$ , i.e., at any time  $t \in [t_0, t_0 + T(n)]$ , the number of idle servers is greater than  $U(t)$ . We prove the claim by induction. Let the claim hold for time  $t^-$  and let there be a request at time  $t$  for content  $C_i$ . If this is not the first request for  $C_i$  in  $[t_0, t_0 + T(n)]$ , by the claim, at  $t = t^-$ , there is at least one idle server which can serve this request. In addition, if there is exactly one server which can serve  $C_i$  at  $t^-$ , then the MYOPIC policy replaces the content of some other idle server with  $C_i$ . Since there are more than  $k_1(nT(n))^{1/\beta}$  idle servers and  $U(t) < k_1(nT(n))^{1/\beta}$ , at  $t^+$ , each content type requested in the interval of interest so far, is stored on at least one currently idle server. Therefore, conditioned on  $F_1$  and  $F_3$ , every request for a particular content type except the first request, is not deferred.

Hence, putting everything together,

$$\mathbb{P}(F_4) \geq 1 - \mathbb{P}(F_1^c) - \mathbb{P}(F_3^c),$$

thus  $\mathbb{P}(F_4) \rightarrow 1$  as  $n \rightarrow \infty$  and the result follows.  $\square$

## D.4 Proof of Theorem 19

We first present an outline of the proof of Theorem 19.

1. Since we are studying the performance of the MYOPIC policy for the Continuous Change Model, the relative order of popularity of contents keeps changing in the interval of interest. We show that w.h.p., the number of content types which are in the  $n^{1/\beta}$  most popular content types at least once in the interval of interest is  $O(n^{1/\beta})$  (Lemma 36).
2. Next, we show that w.h.p., in the interval of interest of length  $b$ , only  $O(n^{1/\beta})$  content types are requested (Lemma 37).
3. By Lemma 34 and the proof of Theorem 18, we know that, conditioned on Step 3, the MYOPIC storage policy ensures that in the interval of interest, once a content type is requested for the first time, there is always at least one idle server which can serve an incoming request for that content. Using this, we conclude that, in the interval of interest, only the first request for a particular content type will be deferred. The proof of Theorem 19 then follows from Step 2.

*Lemma 36.* Let  $G_1$  be the event that, in the interval of interest of length  $b$ , the number of times that a content among the current top  $n^{1/\beta}$  most popular contents changes its position in the popularity ranking is at most  $\frac{4b}{\alpha}n^{1/\beta}\nu$ . Then,  $P(G_1) \geq 1 - o(\frac{1}{n})$ .

*Proof.* The expected number of clock ticks in  $b$  time-units is  $bn\nu$ . The probability that a change in arrival process involves at least one of the current  $n^{1/\beta}$  most popular contents is  $\frac{n^{1/\beta}}{\alpha n}$ . Therefore, the expected number of changes in arrival process which involve at least one of the current  $n^{1/\beta}$  most popular contents is  $\frac{2b\nu}{\alpha}n^{1/\beta}$ . By the Chernoff bound, we have that  $P(G_1) \geq 1 - o(\frac{1}{n})$ .  $\square$

*Lemma 37.* Let  $G_2$  be the event that in the interval of interest, no more than  $O(n^{1/\beta})$  different types of contents are requested. Then,  $\mathbb{P}(G_2^c) = o(\frac{1}{n})$ .

*Proof.* Conditioned on the event  $G_1$  defined in Lemma 36, we have that in the interval of interest, at most  $(\frac{2b}{\alpha}\nu + 1)n^{1/\beta}$  different contents are among the top  $n^{1/\beta}$  most popular contents. Given this, the proof follows the same lines of arguments as in the proof of Lemma 35.  $\square$

The proof of the theorem then follows from Lemma 37 and uses the same line of arguments as in the proof of Theorem 18.s

## D.5 Proof of Theorem 20

To show that GENIE is the optimal policy, we consider the process  $X(t)$  which is the number of occupied servers at time  $t$  when the storage policy is

GENIE. Let  $Y(t)$  be the number of occupied servers at time  $t$  for some other storage policy  $\mathcal{A} \in \mathbb{A}$ . We construct a coupled process  $(X^*(t), Y^*(t))$  such that the marginal rates of change in  $X^*(t)$  and  $Y^*(t)$  is the same as that of  $X(t)$  and  $Y(t)$  respectively.

Recall  $\bar{\lambda} = \frac{\sum_{i=1}^m \lambda_i}{n}$ . At time  $t$ , let  $\mathbf{C}^{\text{GENIE}}(t)$  and  $\mathbf{C}^{\mathcal{A}}(t)$  be the sets of contents stored on idle servers by GENIE and  $\mathcal{A}$  respectively. The construction of the coupled process  $(X^*(t), Y^*(t))$  is described in Figure D.1. We assume that the system starts at time  $t = 0$  and  $X^*(0) = Y^*(0) = 0$ . In this construction, we maintain two counters  $Z_{X^*}$  and  $Z_{Y^*}$  which keep track of the number of departures from the system. Let  $Z_{X^*}(0) = Z_{Y^*}(0) = 0$ . Let  $\text{Exp}(\mu)$  be an Exponential random variable with mean  $\frac{1}{\mu}$  and  $\text{Ber}(p)$  be a Bernoulli random variable which is 1 with probability (w.p.)  $p$ .

---

```

1: Generate:  $\text{ARR} \sim \text{Exp}(n\bar{\lambda})$ ,  $\text{DEP} \sim \text{Exp}(\max\{X^*, Y^*\})$ 
2:  $t = t + \min\{\text{ARR}, \text{DEP}\}$ 
3: if  $\text{ARR} < \text{DEP}$ , then
4:   if  $(X^* = Y^*)$  then
5:     Generate  $u_1 \sim \text{Ber}\left(\frac{\sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i}{n\bar{\lambda}}\right)$ 
6:     if  $(u_1 = 1)$  then
7:        $X^* \leftarrow X^* + 1$ 
8:     Generate  $u_2 \sim \text{Ber}\left(\frac{\sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i}{\sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i}\right)$ 
9:     if  $(u_2 = 1)$  then  $Y^* \leftarrow Y^* + 1$ 
10:    end if
11:  else
12:    Generate  $u_1 \sim \text{Ber}\left(\frac{\sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i}{n\bar{\lambda}}\right)$ 
13:    if  $(u_1 = 1)$  then  $X^* \leftarrow X^* + 1$ 
14:    Generate  $u_2 \sim \text{Ber}\left(\frac{\sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i}{\sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i}\right)$ 
15:    if  $(u_2 = 1)$  then  $Y^* \leftarrow Y^* + 1$ 
16:    end if
17:  else
18:    if  $(X^* \geq Y^*)$  then
19:       $X^* \leftarrow X^* - 1$ ,  $Z_{X^*} \leftarrow Z_{X^*} + 1$ 
20:      Generate  $u_3 \sim \text{Ber}\left(\frac{Y^*}{X^*}\right)$ 
21:      if  $(u_3 = 1)$  then  $Y^* \leftarrow Y^* - 1$ ,  $Z_{Y^*} \leftarrow Z_{Y^*} + 1$ 
22:    else
23:       $Y^* \leftarrow Y^* - 1$ ,  $Z_{Y^*} \leftarrow Z_{Y^*} + 1$ 
24:      Generate  $u_4 \sim \text{Ber}\left(\frac{X^*}{Y^*}\right)$ 
25:      if  $(u_4 = 1)$  then  $X^* \leftarrow X^* - 1$ ,  $Z_{X^*} \leftarrow Z_{X^*} + 1$ 
26:    end if
27:  end if
28: Goto 1

```

---

Figure D.1: Coupled Process



*Lemma 38.*  $X^*(t)$  and  $Y^*(t)$  have the same marginal rates of transition as  $X(t)$  and  $Y(t)$  respectively.

*Proof of Lemma 38.* Consider a small interval of time  $[t_0, t_0 + \delta]$ . By the definition of  $X(t)$ ,

$$\begin{aligned}\mathbb{P}(X(t_0 + \delta) = X(t_0) + 1) &\approx \left( \sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i \right) \delta, \\ \mathbb{P}(X(t_0 + \delta) = X(t_0) - 1) &\approx X(t_0) \delta.\end{aligned}$$

The above probabilities are implicitly conditioned on a suitable state definition for the system; we henceforth drop the conditioning on the state for notational compactness. For the process  $X^*(t)$ ,

$$\begin{aligned}\mathbb{P}(X^*(t_0 + \delta) = X^*(t_0) + 1) &\approx n\bar{\lambda} \left( \frac{\sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i}{n\bar{\lambda}} \right) \delta \\ &= \left( \sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i \right) \delta.\end{aligned}$$

If  $(X^*(t_0) \geq Y^*(t_0))$ ,

$$\mathbb{P}(X^*(t_0 + \delta) = X^*(t_0) - 1) \approx X^*(t_0) \delta,$$

and if  $(X^*(t_0) < Y^*(t_0))$ ,

$$\begin{aligned}\mathbb{P}(X^*(t_0 + \delta) = X^*(t_0) - 1) &\approx Y^*(t_0) \frac{X^*(t_0)}{Y^*(t_0)} \delta \\ &= X^*(t_0) \delta.\end{aligned}$$

The approximations become exact as  $\delta \rightarrow 0$ , since the inter-event (arrival or departure) times are exponential. This proves the lemma for  $X^*$  and  $X$ .

By the definition of  $Y(t)$ ,

$$\begin{aligned}\mathbb{P}(Y(t_0 + \delta) = Y(t_0) + 1) &\approx \left( \sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i \right) \delta, \\ \mathbb{P}(Y(t_0 + \delta) = Y(t_0) - 1) &\approx Y(t_0) \delta.\end{aligned}$$

Consider the case when  $Y^*(t_0) = X^*(t_0)$ .

From Section 5.3.3, we know that, under the GENIE storage policy, if the number of idle servers at time  $t$  is  $k(t)$ , they store the  $k(t)$  most popular contents. Given this, if  $X^*(t_0) = Y^*(t_0)$ ,  $\frac{\sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i}{\sum_{i \in \mathbf{C}^{\text{GENIE}}(t)} \lambda_i} \leq 1$ . Therefore,  $u_2$  as defined in Step 8 of the coupling construction is a valid bernoulli random variable and in addition,  $u_1 \times u_2$  is a bernoulli random variable with parameter  $\left( \frac{\sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i}{n\bar{\lambda}} \right)$ . Therefore, we have that,

$$\begin{aligned}\mathbb{P}(Y^*(t_0 + \delta) = Y^*(t_0) + 1) &\approx n\bar{\lambda} \left( \frac{\sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i}{n\bar{\lambda}} \right) \delta \\ &= \left( \sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i \right) \delta.\end{aligned}$$

If  $Y^*(t_0) \neq X^*(t_0)$ ,

$$\begin{aligned}\mathbb{P}(Y^*(t_0 + \delta) = Y^*(t_0) + 1) &\approx n\bar{\lambda} \left( \frac{\sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i}{n\bar{\lambda}} \right) \delta \\ &= \left( \sum_{i \in \mathbf{C}^{\mathcal{A}}(t)} \lambda_i \right) \delta.\end{aligned}$$

If  $(X^*(t_0) \geq Y^*(t_0))$ ,

$$\begin{aligned}\mathbb{P}(Y^*(t_0 + \delta) = Y^*(t_0) - 1) &\approx X^*(t_0) \frac{Y^*(t_0)}{X^*(t_0)} \delta \\ &= Y^*(t_0) \delta,\end{aligned}$$

and if  $(X^*(t_0) < Y^*(t_0))$ ,

$$\begin{aligned}\mathbb{P}(Y^*(t_0 + \delta) = Y^*(t_0) - 1) &\approx Y^*(t_0)\delta \\ &= Y^*(t_0)\delta.\end{aligned}$$

This completes the proof.  $\square$

*Lemma 39.* Let  $D^{(GENIE)}(t)$  be the number of jobs deferred by time  $t$  by the GENIE adaptive storage policy and  $D^{(\mathcal{A})}(t)$  to be the number of jobs deferred by time  $t$  by a policy  $\mathcal{A} \in \mathbb{A}$ . In the coupled construction, let  $W^*(t)$  be the number of arrivals by time  $t$ . Let,  $D^{X^*}(t) = W^*(t) - Z_{(X^*)}(t) - X^*(t)$  and  $D^{Y^*}(t) = W^*(t) - Z_{(Y^*)}(t) - Y^*(t)$ . Then,  $D^{X^*}(t)$  and  $D^{Y^*}(t)$  have the same marginal rates of transition as  $D^{(GENIE)}(t)$  and  $D^{(\mathcal{A})}(t)$  respectively.

*Proof.* This follows from Lemma 38 due to the fact that  $X(t)$  have the same distribution as  $X^*(t)$  and the marginal rate of increase of  $D^{X^*}(t)$  given  $X^*(t)$  is the same as the rate of increase of  $D^{(GENIE)}(t)$  given  $X(t)$ . The result for  $D^{Y^*}(t)$  follows by the same argument.  $\square$

*Lemma 40.*  $X^* \geq Y^*$  for all  $t$  on every sample path.

*Proof of Lemma 40.* The proof follows by induction.  $X^*(0) = Y^*(0)$  by construction. Let  $X^*(t_0^-) \geq Y^*(t_0^-)$  and let there be an arrival or departure at time  $t_0$ . There are 4 possible cases:

- i: If  $\text{ARR} < \text{DEP}$  and  $X^*(t_0^-) = Y^*(t_0^-)$ ,  $Y^*(t_0) = Y^*(t_0^-) + 1$  only if  $X^*(t_0) = X^*(t_0^-) + 1$ . Therefore,  $X^*(t_0) \geq Y^*(t_0)$ .

- ii: If  $\text{ARR} < \text{DEP}$  and  $X^*(t_0^-) > Y^*(t_0^-)$ ,  $Y^*(t_0) \leq Y^*(t_0^-) + 1 \leq X^*(t_0^-) \leq X^*(t_0)$ . Therefore,  $X^*(t_0) \geq Y^*(t_0)$ .
- iii: If  $\text{DEP} < \text{ARR}$  and  $X^*(t_0^-) = Y^*(t_0^-)$ ,  $X^*(t_0) = Y^*(t_0)$ .
- iv: If  $\text{DEP} < \text{ARR}$  and  $X^*(t_0^-) > Y^*(t_0^-)$ ,  $X^*(t_0) = X^*(t_0^-) - 1 \geq Y^*(t_0^-) \geq Y^*(t_0)$ . Therefore,  $X^*(t_0) \geq Y^*(t_0)$ .

□

*Lemma 41.*  $Z_{X^*} \geq Z_{Y^*}$  for all  $t$  on every sample path.

*Proof.* The proof follows by induction. Since the system starts at time  $t = 0$ ,  $Z_{X^*}(0) = Z_{Y^*}(0)$ . Let  $Z_{X^*}(t_0^-) \geq Z_{Y^*}(t_0^-)$  and let there be a departure at time  $t_0$ . By Lemma 40, we know that,  $X^*(t_0^-) \geq Y^*(t_0^-)$ . Therefore,  $Z_{X^*}(t_0) \geq Z_{Y^*}(t_0)$  by the coupling construction. □

*Proof.* (Proof of Theorem 20)

By Lemmas 40 and 41, for any sample path,

$$X^*(t) + Z_{X^*}(t) \geq Y^*(t) + Z_{Y^*}(t).$$

Therefore, for every sample path, the number of requests already served (not deferred) or being served by the servers by a content delivery system implementing the GENIE policy is more than that by any other storage policy. This implies that for each sample path, the number of requests deferred by GENIE is less than that of any other storage policy. Sample path dominance in the

coupled system implies stochastic dominance of the original process. Using this and Lemma 39, we have that,

$$D^{(GENIE)}(t) \leq_{st} D^{(\mathcal{A})}(t).$$

□

## D.6 Proof of Theorem 21

*Proof of Theorem 21.* The key idea of the GENIE policy is to ensure that at any time  $t$ , if the number of idle servers is  $k(t)$ , the  $k(t)$  most popular contents are stored on exactly one idle server each. Since the total number of servers is  $n$ , and the number of content-types is  $m = \alpha n$  for some constant  $\alpha > 1$ , all content-types  $C_i$  for  $i > n$  are never stored on idle servers by the GENIE policy. This means that under the GENIE policy, all arrivals for content types  $C_i$  for  $i > n$  are deferred. For  $\beta > 1$ , for all  $i, p_i \geq \frac{1}{c_z} i^{-\beta}$ , for some constant  $c_z < \infty$ . The cumulative mass of all content types  $i = n + 1, \dots, \alpha n$  is

$$\begin{aligned} \sum_{i=n+1}^{\alpha n} p_i &\geq \sum_{i=k}^{\alpha n} \frac{1}{c_z} i^{-\beta} \geq \int_{n+1}^{\alpha n+1} \frac{1}{c_z} i^{-\beta} di \\ &\geq \frac{0.9}{c_z(\beta-1)} \frac{1}{(n+1)^{\beta-1}}, \end{aligned}$$

for  $n$  large enough.

Let the length of the interval of interest be  $b$ . The expected number of arrivals of types  $n + 1, n + 2, \dots, \alpha n$ , in the interval of interest is at least  $\frac{0.9b\bar{\lambda}n}{c_z(\beta-1)} \frac{1}{(n+1)^{\beta-1}}$ . Therefore, the expected number of jobs deferred by the

GENIE policy in an interval of length  $b$  is  $\Omega(n^{2-\beta})$ .

□

## D.7 Proof of Theorem 22

*Proof of Theorem 22.* From the proof of Theorem 18, we know that if  $T = o(n^{\beta-1})$ , w.h.p.,

- no more than  $O(nT)^{1/\beta}$  different types of contents are requested,
- once a content  $C_i$  is requested for the first time, the MYOPIC policy ensures that there is always at least 1 idle server which can serve a request for  $C_i$ .

It follows that once a content is requested for the first time, there is at least one copy of that content in the system (more specifically, there is at least one copy of that content on an idle server). Therefore, w.h.p., the number of external fetches is equal to the number of unique content types requested in the interval of interest and the result follows.

For the GENIE policy, before the first arrival, the GENIE policy fetches the  $n$  most popular contents to place on the servers.

Let the number of idle servers at  $t^-$  be  $k(t)$  and let there be a departure from the system at time  $t$ . After this departure, the content of the new idle server is replaced with  $C_{k(t^-)+1}$ . From Lemma 34, we have that with probability  $\geq 1 - o(\frac{1}{n})$ ,  $\Theta(n)$  servers are idle at all times in the interval of interest.

Therefore,  $k(t^-) + 1 > \epsilon n$  for some  $\epsilon > 0$  and  $\lambda_{k(t^-)+1} \leq \frac{\bar{\lambda}n}{(\epsilon n)^\beta}$ . The number of currently busy servers serving a request for content  $k(t^-) + 1$  is stochastically dominated by a Poisson random variable with rate  $\frac{\bar{\lambda}n}{(\epsilon n)^\beta}$ . Therefore, at time  $t^+$ , with probability  $\geq 1 - \frac{\bar{\lambda}n}{(\epsilon n)^\beta}$ , there is no currently busy server in the system serving a request for  $C_{k(t^-)+1}$ . By the properties of the GENIE policy, the other  $k(t^-)$  idle servers store the  $k(t^-)$  most popular contents. Therefore, content  $k(t^-) + 1$  is not available in the system (on a busy or idle server) at time  $t^+$  and will be fetched from the back-end server. Therefore, w.h.p., each departure is followed by an external fetch. Since there are  $\Theta(nT)$  departures in an interval of duration  $T$ , the result follows.

□

# Appendix E

## Proofs from Chapter 6

### E.1 Proof of Theorem 23

It follows from the definition of  $Z_j(T(n))$  that  $Z_j(T(n))$  requests for content  $j$  were being served concurrently at some time in  $[0, T(n)]$ . If only the front-end servers are used to serve these requests, the content  $j$  needs to be replicated on at least  $Z_j(T(n))$  front-end servers at that time. The cost of this alone is  $C_f Z_j(T(n)) + (C_b - C_f)$ , since at least the first copy will have to be fetched from the back-end server at cost  $C_b$  and the other  $(Z_j(T(n)) - 1)$  copies will cost at least  $C_f$  each. If one or more requests are served by the back-end server instead of making a copy on a front-end server, each will cost  $C_m (\geq C_b \geq C_f)$  and therefore, the bound  $C_f Z_j(T(n)) + (C_b - C_f)$  still holds. Summing over all contents requested at least once in the interval  $[0, T(n)]$ , the result follows.

### E.2 Proof of Theorem 24

We first provide an outline of the proof.

- (i) We first bound the quantity  $Z_j(T(n))$  (see Definition 1) for each content  $j$  (Lemmas 42 and 43).



- (ii) Using the bounds obtained, we show that for any  $\delta \in (0, 1 - \bar{\lambda})$ , w.h.p., if  $T(n) = O(n)$ ,  $Z(T(n)) = \sum_{j=1}^n Z_j(T(n)) \leq (1 - \delta)n$  (Lemma 44).
- (iii) It follows from the bound in Step (ii) and the fact that the system begins from the empty state (all servers are idle with no contents on them) that at least  $\delta n$  servers will have no contents on them during  $[0, T(n)]$ . Whenever a new content is fetched, the LRU-R policy first tries to place it on an idle server which has nothing stored on it. Since there is always an empty idle server, once a content is placed on a front-end server, it is not replaced in  $[0, T(n)]$ .
- (iv) Using the property that the LRU-R policy fetches content only when an incoming request cannot be served, we show that the number of fetches by the LRU-R in  $[0, T(n)] = Z(T(n))$ .
- (v) Since every time a content is requested for the first time, it has to be fetched from the back-end server at cost  $C_b$ , we have that the total fetching cost paid by implementing the LRU-R policy is  $C_f Z(T(n)) + (C_b - C_f)U(T(n))$ , which matches the lower bound on the cost of the optimal offline policy (which knows the entire sample path in advance).

*Lemma 42.* Recall the definition of  $X_j(t)$  as in Definition 1. Under Assumption 6.3.1, with probability greater than  $1 - e^{-(\log n)^2}$ ,

$$(i) \quad X_j(t) \leq \lambda_j + \sqrt{\lambda_j} \log n \text{ for } 1 \leq j \leq \frac{n^{1/\beta}}{(\log n)^{6/\beta}},$$

$$(ii) \quad X_j(t) \leq (\log n)^7 \text{ for } \frac{n^{1/\beta}}{(\log n)^{6/\beta}} < j \leq n.$$

*Proof.* Under Assumption 6.3.1,  $\lambda_j = \bar{\lambda} n \frac{j^{-\beta}}{z(\beta)}$ , where  $z(\beta) = \sum_{i=1}^n i^{-\beta}$ , for  $\beta > 2$ . Note that

$$\int_1^\infty x^{-\beta} dx \leq z(\beta) \leq 1 + \int_1^\infty x^{-\beta} dx,$$

therefore,

$$\bar{\lambda} \frac{1}{\beta-1} n j^{-\beta} \leq \lambda_j \leq \bar{\lambda} \frac{\beta}{\beta-1} n j^{-\beta}; \quad 1 \leq j \leq n.$$

Consider an  $M/M/\infty$  queue where the arrival process is Poisson with rate  $\lambda_j$  and the service times are iid  $\text{Exp}(1)$ . Then  $X_j(t)$  is exactly the number of busy servers in the  $M/M/\infty$  queue at time  $t$ , which is well known to be a Poisson random variable with mean  $\lambda_j(1 - e^{-t})$  (see e.g. page 75 of [103]). Hence, for all  $t \geq 0$ ,  $X_j(t) \leq_{st} \text{Poisson}(\lambda_j)$ , where  $\leq_{st}$  denotes stochastic dominance.

For  $j \leq \frac{n^{1/\beta}}{(\log n)^{6/\beta}}$ ,  $\lambda_j = \Omega((\log n)^6)$ . Therefore, using the Chernoff Bound for Poisson random variables, we have that,

$$\begin{aligned} P(X_j(t)) &\geq \lambda_j + \sqrt{\lambda_j} \log n \\ &\leq e^{\sqrt{\lambda_j} \log n} \left( \frac{\lambda_j + \sqrt{\lambda_j} \log n}{\lambda_j} \right)^{-(\lambda_j + \sqrt{\lambda_j} \log n)} \\ &= e^{-(\log n)^2}, \end{aligned}$$

as  $n \rightarrow \infty$ .

For  $j > \frac{n^{1/\beta}}{(\log n)^{6/\beta}}$ ,  $\lambda_j = O((\log n)^6)$ . Therefore, using the Chernoff Bound, we have that,

$$\begin{aligned} P(X_j(t) &\geq (\log n)^7) \\ &\leq e^{(\log n)^7 - (\log n)^6} \left( \frac{(\log n)^7}{(\log n)^6} \right)^{-(\log n)^7} \\ &\leq \left( \frac{\log n}{e} \right)^{-(\log n)^7} \leq e^{-(\log n)^2}, \end{aligned}$$

for  $n$  large enough. □

*Lemma 43.* For a fixed  $\epsilon \in (0, \beta - 2)$ , let  $Y_\epsilon(T(n))$  be the number of requests for contents  $j \geq n^{\frac{1+\epsilon}{\beta-1}}$  in the interval  $[0, T(n)]$ . Then, under Assumption 6.3.1, for  $T = O(n)$ ,  $Y_\epsilon(T(n)) = O(n^{1-\epsilon})$  with probability  $\geq 1 - e^{-(\log n)^2}$ .

*Proof.* Let  $n_2 = n^{\frac{1+\epsilon}{\beta-1}}$ , and  $\lambda^{(n_2)}$  be the cumulative request arrival rate for the contents  $j \in [n_2, n]$ , then

$$\lambda^{(n_2)} = \sum_{n_2 \leq j \leq n} \lambda_j = O(nn_2^{1-\beta}) = O(n^{-\epsilon}).$$

Therefore,  $Y_\epsilon(T(n)) \leq_{st} \text{Poisson}(\lambda^{(n_2)}T(n))$ . By the Chernoff bound, for  $T = O(n)$ ,  $Y_\epsilon(T(n)) = O(n^{1-\epsilon})$  with probability  $\geq 1 - e^{-(\log n)^2}$ . □

*Lemma 44.* Let  $Z_j(T(n)) = \max_{t \in [0, T(n)]} X_j(t)$  and  $Z(T(n)) = \sum_{j=1}^n Z_j(T(n))$ . Under Assumption 6.3.1 for  $\delta \in (0, 1 - \bar{\lambda})$ ,  $Z(T(n)) \leq (1 - \delta)n$  with probability

$$\geq 1 - \frac{1}{n}.$$

*Proof.* For compactness, define

$$n_1 := \frac{n^{1/\beta}}{(\log n)^{6/\beta}}; \quad n_2 := n^{\frac{1+\epsilon}{\beta-1}} \text{ for some } \epsilon \in (0, \beta - 2).$$

Under Assumption 6.3.1, the total request arrival process is  $\text{Poisson}(\bar{\lambda}n)$ . Consider the first  $2\bar{\lambda}T(n)n$  requests after  $t = 0$ . By the union bound and using Lemma 42, with probability  $\geq 1 - 2\bar{\lambda}T(n)ne^{-(\log n)^2}$ , each incoming request sees at most  $\lambda_i + \sqrt{\lambda_i} \log n$  servers serving requests for each content  $i \in [1, n_1]$  and  $(\log n)^7$  servers serving requests for each content  $i \in (n_1, n_2)$ .

Let  $E_1$  be the event that it takes at least  $T(n)$  units of time for the first  $2nT(n)$  content arrivals. By the Chernoff Bound,  $P(E_1^c) \leq e^{-0.386nT(n)}$ .

Therefore, with probability at least  $1 - e^{-0.386nT(n)} - 2\bar{\lambda}T(n)ne^{-(\log n)^2}$ , the following inequalities hold

$$Z_j(T(n)) \leq \lambda_j + \sqrt{\lambda_j} \log n; \text{ for all } j \in [1, n_1], \quad (\text{E.1})$$

$$Z_j(T(n)) \leq (\log n)^7; \text{ for all } j \in (n_1, n_2). \quad (\text{E.2})$$

In addition, from Lemma 43, with probability  $\geq 1 - e^{-(\log n)^2}$ ,

$$\sum_{j=n_2}^n Z_j(T(n)) = O(n^{1-\epsilon}). \quad (\text{E.3})$$

Using (E.1), (E.2) and (E.3), it follows that with probability greater than

$$1 - e^{-0.386nT(n)} - 2\bar{\lambda}T(n)ne^{-(\log n)^2} - e^{-(\log n)^2},$$

$$\begin{aligned} Z(T(n)) &\leq \sum_{i=1}^{n_1} (\lambda_i + \sqrt{\lambda_i} \log n) + \sum_{i=n_1+1}^{n_2} (\log n)^7 \\ &\quad + O(n^{1-\epsilon}) \\ &= \bar{\lambda}n + o(n) \leq (1 - \delta)n, \end{aligned}$$

for any  $\delta \in (0, 1 - \bar{\lambda})$ , by choosing  $n$  large enough,  $\square$

*Proof of Theorem 24.* Let  $E_2$  be the event that  $Z(T(n)) \leq (1 - \delta)n$  for  $\delta \in (0, 1 - \bar{\lambda})$ . By Lemma 44,  $P(E_2) \geq 1 - \frac{1}{n}$  for  $n$  large enough. The rest of this proof is conditioned on  $E_2$ . Since we start from an empty system (no content on front-end servers), conditioned on  $E_2$ :

- No requests are forwarded to the back-end server since there are always at least  $\delta n$  idle servers in the system.
- The LRU-R policy does not replace the content placed on a non-empty front-end server in the interval  $(0, T(n)]$ .

The second point follows because the LRU-R policy replaces the content stored on the front-end servers only if when a request arrives, there is no idle server which can serve the request. At this instant, the LRU-R policy replaces the least recently requested content on idle servers with the content being requested. Let  $R_j(s)$  be the number of front-end servers serving requests for the content  $j$  at time  $s$ ,  $W_j(t) = \max_{s \in [0, t]} R_j(s)$ , and  $W(t) = \sum_{j=1}^n W_j(t)$ . Since we start with an empty system at  $t = 0$ ,  $W(t) \leq Z(T(n)) \leq (1 - \delta)n$ ,

i.e., at each time  $t \in [0, T(n)]$ , there are at least  $\delta n$  servers with no content on them. It is then sufficient to prove that  $C_{LRU-R}(T(n)) = C_f W(T(n)) + (C_b - C_f)U(T(n))$  since we already know from Theorem 23 that  $C_{OPT}(T(n)) \geq C_f Z(T(n)) + (C_b - C_f)U(T(n))$ . The proof is based on using the two properties above and induction on request arrival instances.

First note that the system starts at  $t = 0$  with  $W(0) = U(0) = 0$ . Let  $t_1$  be the instant of the first request arrival. Obviously,  $C_{LRU-R}(t_1^-) = C_f W(t_1^-) + (C_b - C_f)U(t_1^-) = 0$ . To serve the first request, the LRU-R policy will fetch the requested content from the back-end server and replicate it on a front-end server. Therefore,  $C_{LRU-R}(t_1^+) = C_b = C_f W(t_1^+) + (C_b - C_f)U(t_1^+)$ . Next, let  $C_{LRU-R}(t_{r-1}^-) = C_f W(t_{r-1}^-) + (C_b - C_f)U(t_{r-1}^-)$  and  $t_r$  be the instant of arrival of the  $r^{th}$  request. Let the  $r^{th}$  request be for content  $j$ . There are two possible cases:

*Case 1:*  $W_j(t_r^-) = W_j(t_r^+)$ .

In this case, there are currently  $W_j(t_r^-)$  servers with content  $j$  replicated on them and since  $W_j(t_r^-) = W_j(t_r^+)$ , at least one of these servers is idle at  $t_r^-$  and therefore can server the  $r^{th}$  request. Therefore,  $C_{LRU-R}(t_r^+) = C_{LRU-R}(t_r^-) = C_f W(t_r^-) + (C_b - C_f)U(t_r^-) = C_f W(t_r^+) + (C_b - C_f)U(t_r^+)$ .

*Case 2:*  $W_j(t_r^+) = W_j(t_r^-) + 1$ .

This implies that there are  $W_j(t_r^-)$  requests for content  $j$  currently being served by the front-end servers (excluding the  $r^{th}$  request). To serve the  $r^{th}$  request,

content  $j$  needs to be placed on another front-end server.

If this is the first request for  $C_j$ ,  $U(t_r^+) = U(t_r^-) + 1$  and the content needs to be fetched from the back-end server, therefore,  $C_{\text{LRU-R}}(t_r^+) = C_{\text{LRU-R}}(t_r^-) + C_b = C_f(W(t_r^-) + 1) + (C_b - C_f)U(t_r^- + 1) = C_f(W(t_r^+)) + (C_b - C_f)U(t_r^+)$ .

If this is not the first request for  $C_j$ ,  $U(t_r^+) = U(t_r^-)$  and the content will be available on a front-end server. Therefore,  $C_{\text{LRU-R}}(t_r^+) = C_{\text{LRU-R}}(t_r^-) + C_f = C_f(W(t_r^-) + 1) + (C_b - C_f)U(t_r^-) = C_f(W(t_r^+)) + (C_b - C_f)U(t_r^+)$ .

□

### E.3 Proof of Theorem 25

*Proof.* (Proof of Theorem 25) We first show that

$$\mathbb{E}[C_{\text{Learning}}(T(n))] \geq \min\{C_m n, \Gamma_1(T(n))\},$$

for the choice of  $\Gamma_1(T(n))$  as in the statement of Theorem 25.

*Case 1: Phase 1 lasts for  $n$  or more arrivals.*

Since we start with an empty system, no content is stored on the front-end servers in Phase 1. Therefore, the first  $n$  requests are served using the back-end server,  $C_{\text{Learning}}(T(n)) \geq C_m n$ .

*Case 2: Phase 1 lasts for  $n^\gamma$  arrivals for some  $\gamma < 1$ .*

Let  $H_1$  be the event that in  $[0, T(n)]$ , there are no arrivals for content types  $k, k+1, \dots, n$ , for some  $k = O((nT(n))^{\frac{\gamma}{\beta}})$ ,  $0 < \gamma < 1$ . By Lemma 1 of Chapter 5,  $P(H_1) \geq 1 - o(1/n)$ .

Recall the event  $E_2$  defined in the proof of Theorem 24. Conditioned on  $E_2$ , the total number of fetches made by the optimal policy is at most  $n$  and no requests are sent to the back-end server. Using this property, we have that,

$$C_{\text{OPT}}(T(n)) = C_f Z(T(n)) + (C_b - C_f)U(T(n)), \quad (\text{E.4})$$

for  $Z(T(n))$  and  $U(T(n))$  defined in Definition 1.

The rest of the proof is conditioned on  $H_2 = H_1 \cap E_2$ . Let the number of requests forwarded to the back-end server by the learning based policy in  $[0, T(n)]$  be  $D(T(n))$ . By the proof of Theorem 23, the sum of the total number of content fetches and the number of requests forwarded to the back-end server in order to serve all requests which arrive in  $[0, T(n)]$  is at least  $Z(T(n))$ . Since each content fetch costs at least  $C_f$  units and each request served by the back-end server costs  $C_m$  units, we have that,

$$C_{\text{Learning}}(T(n)) \geq C_f Z(T(n)) + (C_m - C_f)D(T(n)). \quad (\text{E.5})$$

From Equations E.4 and E.5, we have that,

$$\begin{aligned} \mathbb{E}[C_{\text{Learning}}(T(n))|H_2] &\geq \mathbb{E}[C_{\text{OPT}}(T(n))|H_2] \\ &\quad + (C_m - C_f)\mathbb{E}[D(T(n))|H_2] \\ &\quad - (C_b - C_f)\mathbb{E}[U(T(n))|H_2]. \end{aligned}$$

For the learning-based static policy, let  $D_1$  be the number of requests sent to the back-end server in Phase 1 and  $D_2$  be the number of requests sent to the



back-end server in Phase 2. Since we start with an empty system, no content is stored on the front-end servers in Phase 1. Therefore,  $\mathbb{E}[D_1|H_2] = n^\gamma$ . By Lemma 4 of Chapter 5,  $\mathbb{E}[D_2|H_2] \geq T(n)n^{1-\gamma+\frac{\gamma}{\beta}}$ . Therefore, minimizing over  $\gamma$ ,  $\mathbb{E}[D(T(n))|H_2] = \Omega(nT(n))^{\frac{1}{2-1/\beta}}$ .

Recall that  $H_1$  is the event that in  $[0, T(n)]$ , there are no arrivals for at least  $n - O((nT(n))^{\frac{\gamma}{\beta}})$  content types. Therefore,  $\mathbb{E}[U(T(n))|H_2] \leq O((nT(n))^{\frac{\gamma}{\beta}})$ , and

$$\begin{aligned} \mathbb{E}[C_{\text{Learning}}(T(n))|H_2] &\geq \mathbb{E}[C_{\text{OPT}}(T(n))|H_2] \\ &\quad + (C_m - C_f)\Omega(nT(n))^{\frac{1}{2-1/\beta}}, \end{aligned}$$

for  $\beta > 1$  and  $\gamma < 1$ . Using the fact that  $H_2$  is a high probability event, as  $n \rightarrow \infty$ , and  $C_{\text{Learning}}(T(n)) \geq C_{\text{OPT}}(T(n))$  by definition, it then follows that,

$$\begin{aligned} \mathbb{E}[C_{\text{Learning}}(T(n))] &\geq \mathbb{E}[C_{\text{OPT}}(T(n))] \\ &\quad + (C_m - C_f)\Omega(nT(n))^{\frac{1}{2-1/\beta}}. \end{aligned}$$

Next we show that  $\mathbb{E}[C_{\text{Learning}}(T(n))] \geq \Gamma_2(T(n))$ , where  $\Gamma_2(T(n)) = C_m\Omega(nT(n))^{\frac{1}{2-1/\beta}}$ . This is immediate because we already showed above that for any learning-based static storage policy,

$$E[D(T(n))] = \Omega(nT(n))^{\frac{1}{2-1/\beta}},$$

as  $n \rightarrow \infty$ . Since  $D(T(n))$  requests are served by the back-end server at cost  $C_m$  each, we have that,

$$\mathbb{E}[C_{\text{Learning}}(T(n))] \geq C_m\Omega(nT(n))^{\frac{1}{2-1/\beta}}.$$

□

## E.4 Proof of Theorem 26

Let  $\mathcal{C}^{(k)}(t)$  be the set of active contents of the top  $k := (n \log n)^{\frac{\epsilon}{\beta-1}}$  most popular classes at time  $t$ , i.e.,  $j \in \mathcal{C}^{(k)}(t)$  means that the content  $j$  is active at time  $t$  and it belongs to a class  $i$ ,  $1 \leq i \leq k$ .

**Definition 2** (Block). We divide time into a sequence of blocks  $B_1, B_2, \dots$ . A block  $B_\ell$  is defined as an interval of time in which  $\mathcal{C}^{(k)}(t)$  remains invariant ( $\mathcal{C}^{(k)}(t) = \mathcal{C}^{(k)}(B_\ell)$  for all  $t \in B_\ell$ ), i.e., the block  $B_\ell$  ends either when the life-span of a content  $j \in \mathcal{C}^{(k)}(B_\ell)$  elapses or a new content of class  $i$ ,  $1 \leq i \leq k$  arrives, which marks the beginning of the  $(\ell + 1)$ -th block.

We first provide an outline of the proof. We prove that under the LRU-R replication policy, the following properties hold w.h.p.

- (i) In each block  $B_\ell$ , every active content in  $\mathcal{C}^{(k)}(B_\ell)$  is requested at least once during  $B_\ell$ .
- (ii) At each time instant  $s$ , the number of idle servers storing copies of active contents in  $\mathcal{C}^{(k)}(s)$  is  $o(n)$ .
- (iii) At each time instant  $s$ , the number of idle servers storing copies of contents (active or otherwise) of class  $i$  such that  $i > k$  is  $o(n)$ .
- (iv) At each request arrival instant, there are  $\Theta(n)$  idle servers.
- (v) The total number of requests for all contents in classes  $\geq k$  over the interval of length  $n^\epsilon$ , where  $\epsilon < \alpha(\beta - 1)$ , is  $o(n)$ .

- (vi) At each request arrival instant, the number of front-end servers storing a particular content  $j$  is  $o(n)$ .

It follows that w.h.p (i)-(vi) hold simultaneously for all the blocks if  $\epsilon < \frac{\beta-1}{2\beta+1}(3\alpha-2+2c)$ . Next we show that conditioned on (i)-(vi), once a content  $j$  in one of the classes  $1 \leq i \leq k$  arrives, no copy of that content on the front-end server is replaced in its life-span.

*Lemma 45.* Consider a block  $B$  and let  $F_1$  be the event that each active content in  $\mathcal{C}^{(k)}(B)$  is requested at least once during  $B$ . Then

$$P(F_1) \geq 1 - \frac{\pi^2}{\gamma_1 2\bar{\lambda}} \frac{k^{\beta+1}}{n^{c+2\alpha-1}} (\log n)^2 - 2n^{-\log n} \rightarrow 1.$$

*Proof.* Let  $F_2$  be the event that the duration of a block is at least  $\frac{(\log n)^2}{\frac{\gamma_2}{\gamma_1} \bar{\lambda} \frac{6}{\pi^2} \frac{n^\alpha}{k^\beta}}$ . By Assumption 6.2.1, the cumulative content arrival of top  $k$  classes to  $\mathcal{C}^{(k)}(B)$  is a Poisson process with rate  $(\gamma_2 \frac{k}{n^\alpha} n^{1-c})$ . Since the life-time of each content is  $\text{Exp}(\gamma_1 n^{-c})$ , we have that  $|\mathcal{C}^{(k)}(B)| \leq_{st} \text{Poisson}(\frac{\gamma_2}{\gamma_1} \frac{k}{n^\alpha} n)$  (similar to  $M/M/\infty$  arguments in the proof of Lemma 42). Therefore, by the Chernoff bound, with probability  $\geq n^{-\log n}$ , the time to first end of the life-span of a content in  $\mathcal{C}^{(k)}(B)$  is  $\leq_{st} \text{Exp}(2\gamma_2 \frac{k}{n^\alpha} n^{1-c})$ . Since the block ends at the arrival of a new content to  $\mathcal{C}^{(k)}(B)$  or the end of the life-span of any content in  $\mathcal{C}^{(k)}(B)$ ,

$$\text{Length of block } B \leq_{st} \text{Exp}\left(3\gamma_2 \frac{k}{n^\alpha} n^{1-c}\right).$$

By the Chernoff bound, we have that,

$$\begin{aligned} P(F_2) &= \exp\left(-3\gamma_2 \frac{k}{n^\alpha} n^{1-c} \frac{(\log n)^2}{\frac{\gamma_2}{\gamma_1} \bar{\lambda} \frac{6}{\pi^2} \frac{n^\alpha}{k^\beta}}\right), \\ &= \exp\left(-3\gamma_1 \frac{\pi^2}{6\bar{\lambda}} k^{\beta+1} n^{1-c-2\alpha} (\log n)^2\right). \end{aligned}$$

The rest of this proof is conditioned on the event  $F_2$ .

Every content  $j \in \mathcal{C}^{(k)}(B)$  has a request arrival rate of at least  $\frac{\gamma_2}{\gamma_1} \bar{\lambda} \frac{6}{\pi^2} \frac{n^\alpha}{k^\beta}$ .

Let  $F_3$  be the event that  $j$  is requested at least once in this interval. Therefore,

$$P(F_3^c) \leq \exp \left( -\frac{\gamma_2}{\gamma_1} \bar{\lambda} \frac{6}{\pi^2} \frac{n^\alpha}{k^\beta} \frac{(\log n)^2}{\frac{\gamma_2}{\gamma_1} \bar{\lambda} \frac{6}{\pi^2} \frac{n^\alpha}{k^\beta}} \right) = n^{-\log n}.$$

On the other hand, by the Chernoff bound, it follows that  $|\mathcal{C}^{(k)}(B)|$  (the number of contents in  $\mathcal{C}^{(k)}(B)$ ) is  $O(n)$  with probability  $\geq 1 - n^{-\log n}$ . Therefore, by the union bound, we have that, for  $n$  large enough,

$$P(F_1^c) \leq \frac{\gamma_1 \pi^2}{2\bar{\lambda}} k^{\beta+1} n^{1-c-2\alpha} (\log n)^2 + 3n^{-\log n}.$$

□

*Lemma 46.* Starting from an empty system, consider the first  $p(n)$  requests, where  $p(n)$  is a polynomial function of  $n$  and  $t_i$  is the arrival time of the  $i$ -th request. Let  $F_4$  be the event that at any arrival instance  $s \in \{t_1, t_1, \dots, t_{p(n)}\}$ , the number of idle servers storing copies of active contents in  $\mathcal{C}^{(k)}(s)$  is  $o(n)$ , where  $k = (n \log n)^{\frac{\epsilon}{\beta-1}}$ . Then,  $P(F_4^c) \leq f(n)e^{-(\log n)^2}$ , where  $f(n) = (p(n) + 1)(\gamma_2/\gamma_1 n + \sqrt{\gamma_2/\gamma_1 n^{1+\alpha}} \log n)$ .

*Proof.* Consider a system consisting of infinite front-end servers. For such a system, once a content is placed on a server, it is never replaced as there are always servers which have nothing stored on them. Consider any time  $t$ ,  $t < t_{p(n)}$ . Consider a content  $j$  of class  $i$  which has arrived to the catalog at time  $\tau$ ,  $\tau \leq t$ , and is still active at time  $t$ . Let  $S_j^{(i)}(t)$  be the number of

requests for this content that have arrived before time  $t$  and have not finished their services yet. Under Assumption 6.2.2,  $S_j^{(i)}(t) \sim \text{Poisson}(\lambda_i(1 - e^{-(t-\tau)}))$ . Under the LRU-R policy, a new copy of a content is replicated on a server only when an incoming request cannot be served by any idle front-end servers. Let  $D_j^{(i)}(t)$  be the number of front-end servers storing a copy of content  $j$  of class  $i$  at time  $t$ . At time  $t$ , under the LRU-R policy,

$$D_j^{(i)}(t) = \max_{s \in [\tau, t]} S_j^{(i)}(s).$$

Let  $I_j^{(i)}(t)$  denote the number of idle servers storing content  $j$  of class  $i$  at time  $t$ . Thus  $I_j^{(i)}(t) = D_j^{(i)}(t) - S_j^{(i)}(t)$ .

**Case 1:**  $i \leq \frac{n^{\alpha/\beta}}{(\log n)^{6/\beta}}$ .

In this case,  $\lambda_i = \Omega((\log n)^6)$ . Therefore, using the Chernoff Bound for Poisson random variables (similar to the proof of Lemma 42) and the union bound over all  $p(n)$  arrivals,

$$\begin{aligned} \mathbb{P}(D_j^{(i)}(t) \geq \lambda_i(1 - e^{-(t-\tau)}) + \sqrt{\lambda_i(1 - e^{-(t-\tau)}) \log n}) & \\ &= p(n)e^{-(\log n)^2}, \\ \mathbb{P}(S_j^{(i)}(t) \leq \lambda_i(1 - e^{-(t-\tau)}) - \sqrt{\lambda_i(1 - e^{-(t-\tau)}) \log n}) & \\ &= e^{-(\log n)^2}, \end{aligned}$$

as  $n \rightarrow \infty$ .

Therefore,  $\mathbb{P}\left(I_j^{(i)}(t) \leq 2\sqrt{\lambda_i(1 - e^{-(t-\tau)}) \log n}\right) \geq 1 - (p(n) + 1)e^{-(\log n)^2}$ .

**Case 2:**  $i > \frac{n^{\alpha/\beta}}{(\log n)^{6/\beta}}$ .

In this case,  $\lambda_i = O((\log n)^6)$ . Therefore, using the Chernoff Bound for Poisson

random variables (similar to the proof of Lemma 42) and the union bound over all  $p(n)$  arrivals,

$$\begin{aligned}
\mathbb{P}(D_j^{(i)}(t) \geq (\log n)^7) &\leq p(n)e^{(\log n)^7 - (\log n)^6} \left( \frac{(\log n)^7}{(\log n)^6} \right)^{-(\log n)^7} \\
&\leq p(n) \left( \frac{\log n}{e} \right)^{-(\log n)^7} \leq p(n)e^{-(\log n)^2},
\end{aligned}$$

as  $n \rightarrow \infty$ .

Therefore,  $\mathbb{P}\left(I_j^{(i)}(t) \leq \log^7 n\right) \geq 1 - p(n)e^{-(\log n)^2}$ .

Let  $Q_i(t)$  be the number of active contents in class  $i$  at time  $t$ . Then  $Q_i(t) \leq_{st} \text{Poisson}\left(\frac{\gamma_2}{\gamma_1}n^{1-\alpha}\right)$ , again by similar  $M/M/\infty$  arguments as in proof of Lemma 1. Therefore,

$$\begin{aligned}
\mathbb{P}\left(Q_i(t) \leq \gamma_2/\gamma_1 n^{1-\alpha} + \sqrt{\gamma_2/\gamma_1 n^{1-\alpha}} \log n\right) &\geq \\
&1 - e^{-(\log n)^2}.
\end{aligned} \tag{E.6}$$

Finally, taking the union bound over all contents, we have that, with probability  $\geq 1 - (p(n) + 1)(\gamma_2/\gamma_1 n + \sqrt{\gamma_2/\gamma_1 n^{1-\alpha}} \log n)e^{-(\log n)^2}$ , the total number of idle servers storing copies of current active contents of classes  $1 \leq i \leq k$ ,  $k = (n \log n)^{\frac{\epsilon}{\beta-1}}$ , is

$$\begin{aligned}
\sum_{i=1}^k \sum_{j=1}^{Q_i(s)} I_j^{(i)}(s) &\leq (\gamma_2/\gamma_1 n^{1-\alpha} + \sqrt{\gamma_2/\gamma_1 n^{1-\alpha}} \log n) \times \\
&\left( \sum_{i=1}^{\frac{n^{\alpha/\beta}}{\log n^{6/\beta}}} \sqrt{\frac{\gamma_2/\gamma_1 n^\alpha}{i^\beta}} \log n + \sum_{i=\frac{n^{\alpha/\beta}}{\log n^{6/\beta}}}^{n^{\frac{\epsilon}{\beta-1}}} (\log n)^7 \right) = o(n),
\end{aligned}$$

for all  $s \in \{t_1, t_1, \dots, t_{p(n)}\}$ .

It can be shown via a coupling argument that for the LRU-R policy, for each content and at each time, the number of idle front-end servers storing the content in the system consisting of  $n$  front-end servers is  $\leq_{st}$  the number of idle front-end servers storing the content in the system consisting of infinite front-end servers. The key idea behind the coupling argument is that no content is ever replaced in the system consisting of infinite front-end server, however, in a system consisting of  $n$  front-end servers, content on idle servers may be replaced if there are no idle servers which have no content on them. Therefore, if we start from empty systems, i.e., no content on any front-end servers, at each step in the coupled sample path, the system with infinite servers has more idle servers storing each content than in the system with  $n$  servers. This completes the proof.  $\square$

*Proof.* (of Theorem 26)

1. We divide the time-interval of length  $T(n) = n^\epsilon$  into blocks (see Definition 2). Let  $G_1$  be the event that the number of blocks in  $T(n)$  is  $6\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c} = O(kn^{\epsilon-\alpha+1-c})$ . From the proof of Lemma 45, we have that the number of blocks in an interval of length  $n^\epsilon \leq_{st}$  Poisson  $(3\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c})$ , by the Chernoff bound, we have that,

$$\begin{aligned} P(G_1^c) &\leq e^{3\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c}} \left( \frac{6\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c}}{3\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c}} \right)^{-6\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c}} \\ &\leq \frac{e^{3\gamma_2 \frac{kn^\epsilon}{n^\alpha} n^{1-c}}}{4} \leq e^{-(\log n)^2}, \end{aligned}$$

for  $n$  large enough.

2. Using (E.6), and union bound over all classes, the total request arrival rate at any time  $t$  is less than  $\bar{\lambda}(n + \sqrt{\gamma_2/\gamma_1 n^{1+\alpha}} \log n)$ . With probability  $\geq 1 - e^{-(\log n)^2}$  the number of content arrivals in an interval of length  $n^\epsilon$  is a polynomial in  $n$ . Let this polynomial be  $p_1(n)$ . Hence, taking the union bound over all the  $p_1(n)$  content arrivals, we have that, the total arrival rate in this interval of length  $n^\epsilon$  is  $\leq \bar{\lambda}(n + \sqrt{\gamma_2/\gamma_1 n^{1+\alpha}} \log n)$  with probability  $\geq 1 - p_1(n)e^{-(\log n)^2}$ . Conditioned on this, the request arrival process  $\leq_{st}$  the Poisson process with rate  $\bar{\lambda}(n + \sqrt{\gamma_2/\gamma_1 n^{1+\alpha}} \log n)$ . Since the service time of each request is a  $\text{Exp}(1)$ , the number of concurrent requests being served at any time is  $\leq_{st}$   $\text{Poisson}(\bar{\lambda}(n + \sqrt{\gamma_2/\gamma_1 n^{1+\alpha}} \log n))$  (similar to the proof of Lemma 42). Since there are  $n$  front-end servers and the number of busy servers at any time  $\leq_{st}$   $\text{Poisson}(\bar{\lambda}(n + \sqrt{\gamma_2/\gamma_1 n^{1+\alpha}} \log n))$  and  $\bar{\lambda} < 1$ , w.h.p. at each request arrival instant, there are  $\Theta(n)$  idle front-end servers. More specifically, by the Chernoff bound for Poisson random variables, at any time-instant,

$$P(\Theta(n) \text{ idle servers}) \geq 1 - e^{-(\log n)^2}.$$

Let  $G_2$  be the event that for the first  $p(n) = \text{poly}(n)$  requests, at least  $\Theta(n)$  servers are free on each request arrival. By the union bound over the first  $p(n)$  request arrivals, we have that,

$$P(G_2^c) \leq (p_1(n) + p(n))e^{-(\log n)^2}.$$



3. Let  $G_3$  be the event that the total request arrivals for all contents in content classes  $\geq k$  for  $k = (n \log n)^{\frac{\epsilon}{\beta-1}}$  in the interval of length  $n^\epsilon$  where  $\epsilon < \alpha(\beta - 1)$  is  $o(n)$ . Under Assumptions 6.2.1 and 6.2.2, by the Chernoff bound, we have that,

$$P(G_3^c) \leq e^{-(\log n)^2}.$$

4. Let  $G_4$  be the event that the total request arrivals in the interval of length  $n^\epsilon$  is  $p(n) = \text{poly}(n)$ . Under Assumptions 6.2.1 and 6.2.2, using arguments similar to point (2) above, by the Chernoff bound, we have that,

$$P(G_4^c) \leq e^{-(\log n)^2},$$

for  $n$  large enough.

5. Let  $G_5$  be the event that  $F_1$  happens for the first  $L = O(kn^{\epsilon-\alpha+1-c})$  blocks.
6. Let  $G_6$  be the event that  $D_j^{(i)}$  as defined in Lemma 46 is  $o(n)$  for a content  $j$  in class  $i$  for the first  $p(n)$  requests. From the proof of Lemma 46, we have that,

$$\begin{aligned} P(D_j^{(i)}(t) \geq \lambda_i(1 - e^{-(t-\tau)}) + \sqrt{\lambda_i(1 - e^{-(t-\tau)}) \log n}) \\ = p(n)e^{-(\log n)^2}, \end{aligned}$$

where  $p(n)$  is a polynomial function of  $n$ . Since  $\lambda_i = O(n^\alpha)$  for  $\alpha < 1$ , it follows that,

$$P(G_6^c) \leq p(n)e^{-(\log n)^2}.$$

Let  $G = G_1 \cap G_2 \cap G_3 \cap G_4 \cap G_5 \cap G_6 \cap F_4$ . Using the union bound, we have that,

$$\begin{aligned} P(G^c) &\leq (2p(n) + p_1(n) + f(n) + 5 + 3L)e^{-(\log n)^2} \\ &\quad + L \frac{\gamma_1 \pi^2}{2\lambda} k^{\beta+1} n^{1-c-2\alpha} (\log n)^2, \end{aligned}$$

where  $f(n)$  is a polynomial function of  $n$  as defined in Lemma 46. For  $\epsilon < \frac{\beta-1}{2\beta+1}(3\alpha - 2 + 2c)$  and  $k = (n^\epsilon \log n)^{\frac{1}{\beta-1}}$ ,  $P(G) \geq 1 - o(1)$ . The rest of this proof is conditioned on  $G$ .

Next we show that conditioned on  $G$ , once a content  $j$  in one of the classes  $1 \leq i \leq k$  arrives, no copy of that content on the front-end server is replaced in its life-span. Let the current block be  $B_\ell$ . We first introduce some notation:

- $\xi_1(B_\ell) = \mathcal{C}^{(k)}(B_{\ell-1}) \setminus \mathcal{C}^{(k)}(B_\ell)$ : The content in classes  $\leq k$  whose life-span elapsed at the beginning of block  $\ell$ . Note that  $\xi_1(B_\ell) = \emptyset$  if block  $\ell$  started due to the arrival of a new content in class  $\leq k$ .
- $\xi_2(B_\ell) = \mathcal{C}^{(k)}(B_\ell) \setminus \mathcal{C}^{(k)}(B_{\ell-1})$ : The arrival to classes  $\leq k$  at the beginning of block  $\ell$ . Note that  $\xi_2(B_\ell) = \emptyset$  if block  $\ell$  started due to the end of the life-span of a content in class  $\leq k$  that was active in  $B_{\ell-1}$ .

- $U_k(t)$ : Set of all contents (active or otherwise) in classes  $> k$ .
- $I_j(t)$ : Number of idle servers storing content  $j$  at time  $t$ .

Conditioned on (i)-(vi), at any time  $t \in B_\ell$ ,

$$\sum_{j \in \mathcal{C}^{(k)}(B_{\ell-1})} I_j(t) + \sum_{j \in U_k} I_j(t) + I_{\xi_1(B_\ell)}(t) + I_{\xi_2(B_\ell)}(t) = o(n)$$

and there are  $\Theta(n)$  idle servers. Therefore, for  $n$  large enough, at any  $t \in B_\ell$ , there are idle servers storing either nothing, or contents other than contents in  $\mathcal{C}^{(k)}(B_{\ell-1}) \cup U_k(t) \cup \xi_1(B_\ell) \cup \xi_2(B_\ell)$ . Conditioned on (i)-(v) (specifically (i)), all contents in  $\mathcal{C}^{(k)}(B_{\ell-1})$  were requested at least once in the previous block. Given this, contents in  $\mathcal{C}^{(k)}(B_{\ell-1})$  are more recently requested than then contents in  $\cup_{r \leq \ell-1} \xi_1(B_r)$  (i.e. contents of the top  $k$  classes whose life-span elapsed before the beginning of the  $(\ell - 1)$ -th block). Therefore, if a new copy of a content needs to be placed on an idle server in this block, the LRU-R policy will pick a server which does not store a copy of a content in  $\mathcal{C}^{(k)}(B_{\ell-1})$ . Therefore, once a content in  $\mathcal{C}^{(k)}(B_\ell)$  arrives, no copy of that content on a front-end server is replaced during its life-span.

Given this, since the LRU-R policy creates a new copy of a content only if all the front-end servers storing that content are busy, at time  $t$ , the LRU-R policy would have replicated content  $j$  in class  $i \leq k$  on the minimum number of copies required to serve all the requests for content  $j$  which arrive before time  $t$ . Therefore, only requests for contents in classes  $> k$  can potentially be served at a higher cost ( $\leq C_m$ ) by the LRU-R policy as compared to the

optimal storage policy. Conditioned on (i)-(vi) (specifically (v)), the total number of such requests in  $T(n) = n^\epsilon$  is  $o(n)$ . Therefore, we have that,  $C_{\text{LRU-R}}(T(n)) = C_{\text{OPT}}(T(n)) + C_m o(n)$ . Since  $C_{\text{OPT}}(T(n)) = \Omega(n)$  w.h.p. if  $T(n) = \Omega(1)$ , the result follows.

□

## Bibliography

- [1]
- [2] 3gpp tr 25.913. requirements for evolved utra (e-utra) and evolved utran (e-utran). March, 2006.
- [3] G. Aggarwal, G. Goel, C. Karande, and A. Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. *CoRR*, 2010.
- [4] M. Ahmed, S. Spagna, F. Huici, and S. Niccolini. A peek into the future: predicting the evolution of popularity in user generated content. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 607–616. ACM, 2013.
- [5] M. Ahmed, S. Traverso, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today’s content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, October 2013.
- [6] M. Ahmed, S. Traverso, P. Giaccone, E. Leonardi, and S. Niccolini. Analyzing the performance of lru caches under non-stationary traffic patterns. *arXiv preprint arXiv:1301.4909*, 2013.

- [7] J. Andrews. Seven ways that hetnets are a cellular paradigm shift. *IEEE Communications Magazine*, 51(3), 2013.
- [8] J. Andrews, H. Claussen, M. Dohler, S. Rangan, and M. Reed. Femtocells: Past, present, and future. *Selected Areas in Communications, IEEE Journal on*, 30(3):497–508, 2012.
- [9] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan. Optimal content placement for a large-scale VoD system. In *Proceedings of ACM CoNEXT*, New York, NY, USA, 2010.
- [10] P. Baptiste, P. Brucker, S. Knust, and V. Timkovsky. Ten notes on equal-processing-time scheduling. *4OR: A Quarterly Journal of Operations Research*, 2:111–127, 2004.
- [11] A. Barbieri, P. Gaal, S. Geirhofer, T. Ji, D. Malladi, Y. Wei, and F. Xue. Coordinated downlink multi-point communications in heterogeneous cellular networks. In *Information Theory and Applications Workshop (ITA), 2012*, pages 7–16. IEEE, 2012.
- [12] P. Billingsley. *Probability and Measure*. Wiley,, 1995.
- [13] B. Birnbaum and C. Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, March 2008.
- [14] S. Bodas, S. Shakkottai, L. Ying, and R. Srikant. Scheduling in multi-channel wireless networks: Rate function optimality in the small buffer regime. In *Proceedings of SIGMETRICS/performance Conf.*, 2009.

- [15] S. Bodas, S. Shakkottai, L. Ying, and R. Srikant. Low-complexity scheduling algorithms for multi-channel downlink wireless networks. In *Proceedings of IEEE Infocom*, 2010.
- [16] S. Bodas, S. Shakkottai, L. Ying, and R. Srikant. Scheduling for small delay in multi-rate multi-channel wireless networks. In *Proceedings of IEEE Infocom*, 2011.
- [17] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99*, pages 126–134, 1999.
- [18] L. Bui, R. Srikant, and A. Stolyar. A novel architecture for reduction of delay and queueing structure complexity in the back-pressure algorithm. *IEEE/ACM Trans. Network.*, 19(6):1597–1609, 2011.
- [19] S. Chen, L. Tong, and T. He. Optimal deadline scheduling with commitment. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 111–118, Sept. 2011.
- [20] W. Cheung, T. Quek, and M. Kountouris. Throughput optimization, spectrum allocation, and access control in two-tier femtocell networks. *Selected Areas in Communications, IEEE Journal on*, 30(3):561–574, 2012.
- [21] M. Chrobak, W. Jawor, J. Sgall, and T. Tich. Online scheduling of equal-length jobs: Randomization and restarts help. In Josep Daz,

- Juhani Karhumki, Arto Lepist, and Donald Sannella, editors, *Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 145–156. Springer Berlin / Heidelberg, 2004.
- [22] Cisco Whitepaper: [http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white\\_paper\\_c11-481360.html](http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html).
- [23] D. Ciullo, V. Martina, M. Garetto, E. Leonardi, and G.L. Torrisi. Stochastic analysis of self-sustainability in peer-assisted VoD systems. In *IEEE INFOCOM*, pages 1539–1547, 2012.
- [24] E. Coffman and P. Denning. *Operating Systems Theory*. Prentice-Hall, 1973.
- [25] T. Cover, M. Thomas, and J. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
- [26] J. Ding and G. Zhang. Online scheduling with hard deadlines on parallel machines. In Siu-Wing Cheng and Chung Poon, editors, *Algorithmic Aspects in Information and Management*, volume 4041 of *Lecture Notes in Computer Science*, pages 32–42. Springer Berlin / Heidelberg, 2006.
- [27] A. Dua and N. Bambos. Downlink wireless packet scheduling with deadlines. *Mobile Computing, IEEE Transactions on*, 6(12):1410–1425, dec. 2007.
- [28] T. Ebenlendr and J. Sgall. A lower bound for scheduling of unit jobs with immediate decision on parallel machines. In Evripidis Bampis and



Martin Skutella, editors, *Approximation and Online Algorithms*, volume 5426 of *Lecture Notes in Computer Science*, pages 43–52. Springer Berlin / Heidelberg, 2009.

- [29] A. Eryilmaz, R. Srikant, and J. Perkins. Stable scheduling policies for fading wireless channels. *IEEE/ACM Trans. Network.*, 13:411–424, April 2005.
- [30] W. A. Gale. Good-turing smoothing without tears. *Journal of Quantitative Linguistics*, 2:217–237, 1995.
- [31] L. Georgiadis, M. J. Neely, and L. Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1(1), 2006.
- [32] L. Georgiadis, M.J. Neely, and L. Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1(1):1–144, 2006.
- [33] D. Gesbert, S. Hanly, H. Huang, S. Shamaï, O. Simeone, and W. Yu. Multi-cell mimo cooperative networks: A new look at interference. *Selected Areas in Communications, IEEE Journal on*, 28(9):1380–1408, 2010.
- [34] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube traffic characterization: A view from the edge. In *7th ACM SIGCOMM Conference on Internet Measurement*, pages 15–28, 2007.

- [35] G. Goel and A. Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 982–991, Philadelphia, PA, USA, 2008.
- [36] S.A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34(2):370 – 389, 2000.
- [37] M.H. Goldwasser and M. Pedigo. Online nonpreemptive scheduling of equal-length jobs on two identical machines. *ACM Trans. Algorithms*, 5(1):2:1–2:18, December 2008.
- [38] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.
- [39] A. Gopalan, C. Caramanis, and S. Shakkottai. On the value of coordination and delayed queue information in multicellular scheduling. *Automatic Control, IEEE Transactions on*, 58(6):1443–1456, 2013.
- [40] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2), Mar. 2000.
- [41] [http://www.ericsson.com/news/120223\\_it\\_comes\\_back\\_to\\_backhaul\\_244159020\\_c](http://www.ericsson.com/news/120223_it_comes_back_to_backhaul_244159020_c).
- [42] I-H. Hou and P.R. Kumar. Queueing systems with hard delay constraints: a framework for real-time communication over unreliable wireless channels. *Queueing Syst. Theory Appl.*, 71(1-2):151–177, June 2012.

- [43] A. Iamnitchi, M. Ripeanu, and I. Foster. Small-world file-sharing communities. In *IEEE INFOCOM*, March 2004.
- [44] R. Irmer, H. Droste, P. Marsch, M. Grieger, G. Fettweis, S. Brueck, H. Mayer, L. Thiele, and V. Jungnickel. Coordinated multipoint: Concepts, performance, and field trial results. *Communications Magazine, IEEE*, 49(2):102–111, 2011.
- [45] J.J. Jaramillo, R. Srikant, and L. Ying. Scheduling for optimal rate allocation in ad hoc networks with heterogeneous delay constraints. *Selected Areas in Communications, IEEE Journal on*, 29(5):979–987, may 2011.
- [46] B. Ji, G. R. Gupta, X. Lin, and N. B. Shroff. Performance of low-complexity greedy scheduling policies in multi-channel wireless networks: Optimal throughput and near-optimal delay. In *Proceedings of IEEE INFOCOM*, 2013.
- [47] B. Ji, C. Joo, and N. Shroff. Throughput-optimal scheduling in multi-hop wireless networks without per-flow information. In *Proceedings of WiOPT*, 2011.
- [48] Bo Ji, Gagan R Gupta, Manu Sharma, Xiaojun Lin, and Ness B Shroff. Achieving optimal throughput and near-optimal asymptotic delay performance in multi-channel wireless networks with low complexity: A practical greedy scheduling policy. *arXiv preprint arXiv:1212.1638*, 2012.

- [49] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. *Kluwer International Series in Engineering and Computer Science*, pages 153–179, 1996.
- [50] B. Kalyanasundaram and K.R. Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233:2000, 2000.
- [51] J. Kangasharju, K.W. Ross, and D.A. Turner. Optimizing file availability in peer-to-peer content distribution. In *INFOCOM*, 2007.
- [52] J. Kangasharjua, J. Roberts, and K.W. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25:376–383, 2002.
- [53] R.M. Karp, U.V. Vazirani, and V.V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of Computing*, Baltimore, Maryland, May 1990.
- [54] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: Analog network coding. *SIGCOMM Comput. Commun. Rev.*, 37(4):397–408, August 2007.
- [55] F . Khan, Z . Pi, and S. Rajagopal. Millimeter-wave mobile broadband with large scale spatial processing for 5g mobile communication. In

*Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 1517–1523. IEEE, 2012.

- [56] T.G. Kurtz. Solutions of ordinary differential equations as limits of pure jump markov processes. *Journal of Applied Probability*, 7(1):pp. 49–58, 1970.
- [57] J. Laneman and G. Wornell. Distributed space-time-coded protocols for exploiting cooperative diversity in wireless networks. *Information Theory, IEEE Transactions on*, 49(10):2415–2425, 2003.
- [58] M. Leconte, M. Lelarge, and L. Massoulie. Bipartite graph structures for efficient balancing of heterogeneous loads. In *the 12th ACM SIGMETRICS Conference*, pages 41–52, 2012.
- [59] M. Leconte, M. Lelarge, and L. Massoulie. Adaptive replication in distributed content delivery networks. *Preprint*, 2013.
- [60] Jae-Ha Lee. Online deadline scheduling: multiple machines and randomization. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '03, pages 19–23, New York, NY, USA, 2003. ACM.
- [61] X. Lin, R. Ganti, P. Fleming, and J. Andrews. Towards understanding the fundamentals of mobility in cellular networks. 2012.

- [62] S. Liu, E. Ekici, and L. Ying. Scheduling in multihop wireless networks without back-pressure. In *Annual Conference on Communication, Control and Computing (Allerton)*, 2010.
- [63] A. Lozano, R. Heath Jr, and J. Andrews. Fundamental limits of cooperation. Arxiv: CoRR abs/1204.0011, 2012.
- [64] <http://www.3gpp.org/lte-advanced>.
- [65] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *16th international conference on Supercomputing*, 2002.
- [66] A.D. McAllester and R.E. Schapire. On the convergence rate of Good-Turing estimators. In *COLT Conference*, pages 1 – 6, 2000.
- [67] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized on-line matching. In *Proceedings of FOCS*, 2005.
- [68] S. Moharir, J. Ghaderi, S. Sanghavi, and S. Shakkottai. Serving content with unknown demand: the high-dimensional regime. In *the 14th ACM SIGMETRICS Conference*, 2014.
- [69] S. Moharir and S. Sanghavi. Online load balancing and correlated randomness. In *Annual Conference on Communication, Control and Computing (Allerton)*, 2012.

- [70] S. Moharir, S. Sanghavi, and S. Shakkottai. Online load balancing under graph constraints. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, pages 363–364. ACM, 2013.
- [71] S. Moharir and S. Shakkottai. Maxweight vs backpressure: Routing and scheduling for multi-channel relay networks. In *Proceedings of IEEE Infocom*, Turin, Italy, April 2013.
- [72] S. Moharir and S. Shakkottai. Maxweight vs backpressure: Routing and scheduling for multi-channel relay networks. Technical report, 2014.
- [73] L. Qingwen and W. Xin and G.B. Giannakis. A cross-layer scheduling algorithm with qos support in wireless networks. *Vehicular Technology, IEEE Transactions on*, 55(3):839–847, may 2006.
- [74] M. Neely, E. Modiano, and C. Rohrs. Dynamic power allocation and routing for time-varying wireless networks. *IEEE J. Sel. Areas Commun.*, 23(1):89–103, 2005.
- [75] [www.netflix.com](http://www.netflix.com).
- [76] A. Nosratinia, T. Hunter, and A. Hedayat. Cooperative communication in wireless networks. *Communications Magazine, IEEE*, 42(10):74–80, 2004.

- [77] F. Olmos, B. Kauffmann, A. Simonian, and Y. Carlinet. Catalog dynamics: Impact of content publishing and perishing on the performance of a lru cache. *arXiv preprint arXiv:1403.5479*, 2014.
- [78] A. Ozgur, O. Leveque, and D. Tse. Hierarchical cooperation achieves optimal capacity scaling in ad hoc networks. *Information Theory, IEEE Transactions on*, 53(10):3549–3572, 2007.
- [79] Z. Pi and F. Khan. An introduction to millimeter-wave mobile broadband systems. *Communications Magazine, IEEE*, 49(6):101–107, 2011.
- [80] V. Raghunathan, V. Borkar, M. Cao, and P.R. Kumar. Index policies for real-time multicast scheduling for wireless broadcast systems. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1570 –1578, april 2008.
- [81] S. Rajagopal, S. Abu-Surra, Z. Pi, and F. Khan. Antenna array design for multi-gbps mmwave mobile broadband communication. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6. IEEE, 2011.
- [82] I. Rhee, M. Shin, S. Hong, K. Lee, S. Kim, and S. Chong. On the levy-walk nature of human mobility. *IEEE/ACM Transactions on Networking (TON)*, 19(3):630–643, 2011.
- [83] S. Shakkottai. Effective capacity and qos for wireless scheduling. *IEEE Trans. Automat. Contr.*, 53(3):749–761, 2008.



- [84] S. Shakkottai and R. Srikant. Scheduling real-time traffic with deadlines over a wireless channel. *Wireless Networks*, 8(1):13–26, January 2002.
- [85] M. Sharma and X. Lin. Ofdm downlink scheduling for delay-optimality: Many-channel many-source asymptotics with general arrival processes. In *Proceedings of ITA*, 2011.
- [86] LTE Small Cells, [http://http://en.wikipedia.org/wiki/Small\\_cell](http://en.wikipedia.org/wiki/Small_cell).
- [87] A. Stolyar. Large deviations of queues sharing a randomly time-varying server. *Queueing Systems*, 59(2):1–35, 2008.
- [88] A. Stolyar. Large number of queues in tandem: Scaling properties under back-pressure algorithm. *Queueing Systems*, 67(2):111–126, 2011.
- [89] B. Tan and L. Massoulié. Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM Trans. Networking*, 21:566–579, 2013.
- [90] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. Automat. Contr.*, 37(12):1936–1948, 1992.
- [91] L. Tassiulas and A. Ephremides. Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Trans. Automat. Contr.*, 39:466–478, 1993.

- [92] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today’s content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, 2013.
- [93] J.N. Tsitsiklis and K. Xu. Queueing system topologies with limited flexibility. In *SIGMETRICS ’13*, 2013.
- [94] G. Valiant and P. Valiant. Estimating the unseen: An  $n/\log(n)$ -sample estimator for entropy and support size, shown optimal via new clts. In *Proceedings of the 43rd annual ACM Symposium on Theory of Computing*, pages 685–694, 2011.
- [95] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 117–130, 2002.
- [96] V. Venkataramanan and X. Lin. Structural properties of ldp for queue-length based wireless scheduling algorithms. In *Annual Conference on Communication, Control and Computing (Allerton)*, 2007.
- [97] V. Venkataramanan, X. Lin, L. Ying, and S. Shakkottai. On scheduling for minimizing end-to-end buffer usage over multihop wireless networks. In *Proceedings of IEEE Infocom*, 2010.

- [98] A. Viterbi, A. Viterbi, K. Gilhousen, and E. Zehavi. Soft handoff extends cdma cell coverage and increases reverse link capacity. *Selected Areas in Communications, IEEE Journal on*, 12(8):1281–1288, 1994.
- [99] R. B. Wallace and W. Whitt. A staffing algorithm for call centers with skill-based routing. *Manufacturing and Service Operations Management*, 7:276–294, 2007.
- [100] J. Wang. A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29:36–46, 1999.
- [101] S. Williams, M. Abrams, C.R. Standridge, G. Abdulla, and E.A. Fox. Removal policies in network caches for world-wide web documents. In *SIGCOMM’96*, 1996.
- [102] S. Williams, M. Abrams, C.R. Standridge, G. Abdulla, and E.A. Fox. Caching proxies: limitations and potentials. In *the 4th International WWW Conference*, December 1995.
- [103] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall; 1st edition, 1989.
- [104] D. Wong and T. Lim. Soft handoffs in cdma mobile systems. *Personal Communications, IEEE*, 4(6):6–17, 1997.
- [105] W. Wu and J.C.S. Lui. Exploring the optimal replication strategy in P2P-VoD systems: Characterization and evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 23, August 2012.

- [106] A.C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 222 –227, 31 1977-nov. 2 1977.
- [107] L. Ying, S. Shakkottai, and A. Reddy. On combining shortest-path and back-pressure routing over multihop wireless networks. In *Proceedings of IEEE Infocom*, 2009.
- [108] L. Ying, R. Srikant, A. Eryilmaz, and G. Dullrud. A large deviations analysis of scheduling in wireless networks. *IEEE Trans. Inform. Theory*, 52(11):5088–5098, 2006.
- [109] [www.youtube.com](http://www.youtube.com).
- [110] H. Yu, D. Zheng, B.Y. Zhao, and W. Zheng. Understanding user behavior in large scale video-on-demand systems. In *EuroSys*, April 2006.
- [111] X. Zhou and C. Xu. Optimal video replication and placement on a cluster of video-on-demand servers. In *International Conference on Parallel Processing*, pages 547–555, 2002.
- [112] Y. Zhou, T.Z. Fu, and D.M. Chiu. On replication algorithm in P2P-VoD. *IEEE/ACM Transactions on Networking*, pages 233 – 243, 2013.
- [113] W. Zhuang and M. Ismail. Cooperation in wireless communication networks. *Wireless Communications, IEEE*, 19(2):10–20, 2012.
- [114] Zipf’s Law on Wikipedia: [http://en.wikipedia.org/wiki/Zipf’s\\_law](http://en.wikipedia.org/wiki/Zipf's_law).

## Vita

Sharayu Arun Moharir received the Master of Technology degree in Communication and Signal Processing and the Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology (IIT), Bombay in 2009. Her doctoral research at the University of Texas at Austin has been co-advised by Prof. Sujay Sanghavi and Prof. Sanjay Shakkottai.

Permanent address: sharayu.moharir@gmail.com

This dissertation was typeset with L<sup>A</sup>T<sub>E</sub>X<sup>†</sup> by the author.

---

<sup>†</sup>L<sup>A</sup>T<sub>E</sub>X is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's T<sub>E</sub>X Program.